

# PROCESS MINING

## Lab Session 5

### Hands-on Session on Conformance Checking in ProM

#### Content

Laboratory Session .....	2
1. Computing the Alignments.....	4
A. Experience on a First Event Log related to the Repair Process .....	4
B. Experience on a Second Event Log related to the Repair Process .....	7
2. Precision and Generalization of Process Models .....	10
3. A real-life example: The road-traffic fine management process .....	11
4. Exercise .....	15

## Laboratory Session

This hands-on session starts from the model obtained for the *Inductive Miner* for the Repair Example using the log in file *repairExample.xes* (see Section 2 of the document related to the laboratory session on Process Discovery for more. First, choose action *Mine with Inductive visual Miner*, then *export model* from Show features as *Petri Net* and obtain Figure 1).

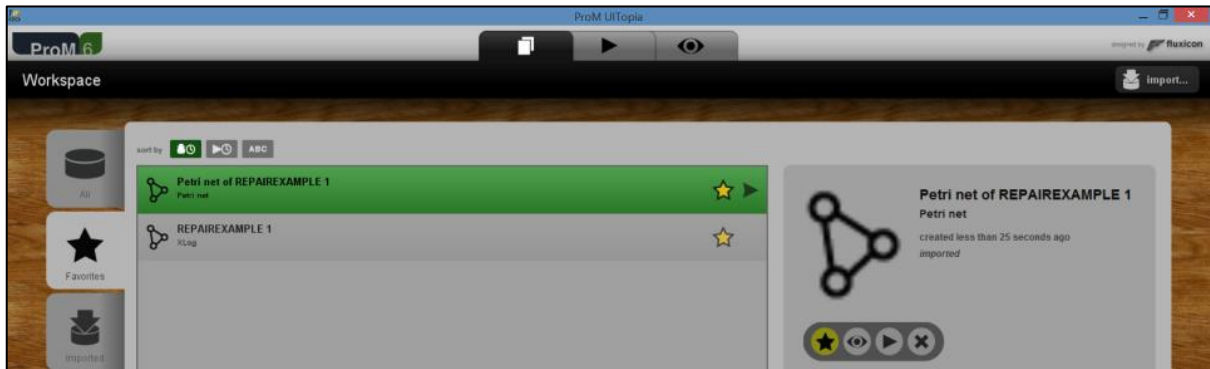


Figure 1. Creating Petri net with the Inductive Miner

The discovered model contains one transition for each activity. The conformance checker expects that each transition is associated with one event type, namely with one combination of activity name life cycle transition. Therefore, we need to filter out the start events from *repairExample.xes* to obtain reliable results while retaining every complete event. This is achieved as usual through the *Filter log using Simple Heuristics* plug-in, and Figure 2 shows the suitable configuration for the case in question.

2

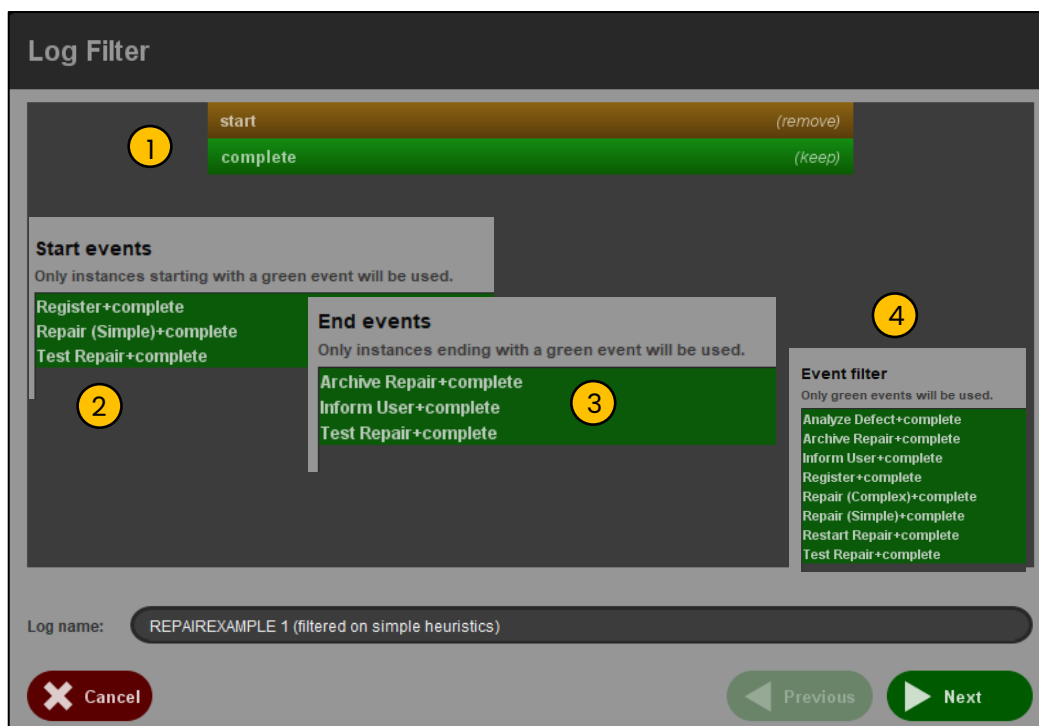


Figure 2. Filtering start event to avoid self loop by Filter log using Simple Heuristic

Once the event log is filtered, alignments are computed between the traces in the *filtered event log* and the *Inductive Miner model* shown in Figure 3.

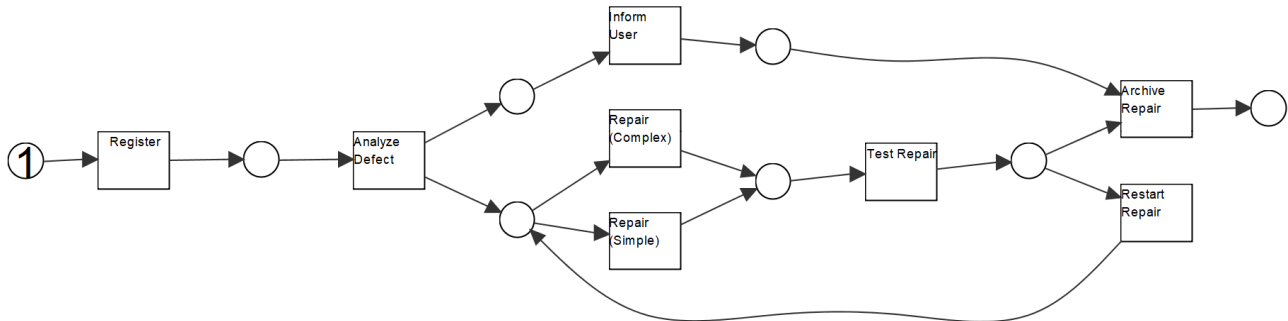


Figure 3. Created Petri net

Due to a bug in ProM, before applying the conformance checking, make sure the Petri Net is visualized at least once (i.e. the "Eye" button is pressed at least once). Otherwise, you might encounter a Java Exception (i.e. an internal error) of type *NullPointerException* like in Figure 4.

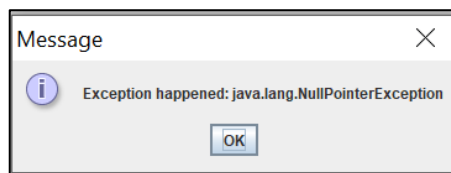


Figure 4. Java exception error

## 1. Computing the Alignments

### A. Experience on a First Event Log related to the Repair Process

This laboratory session illustrates how to use the *alignment-based conformance checker* in ProM to compute fitness, precision, and generalization.

For the *alignment-based conformance checking*, we implement a plug-in *Replay a Log on Petri Net for Conformance Analysis* in ProM. This plug-in takes a *Petri net and log as input* like in Figure 5.

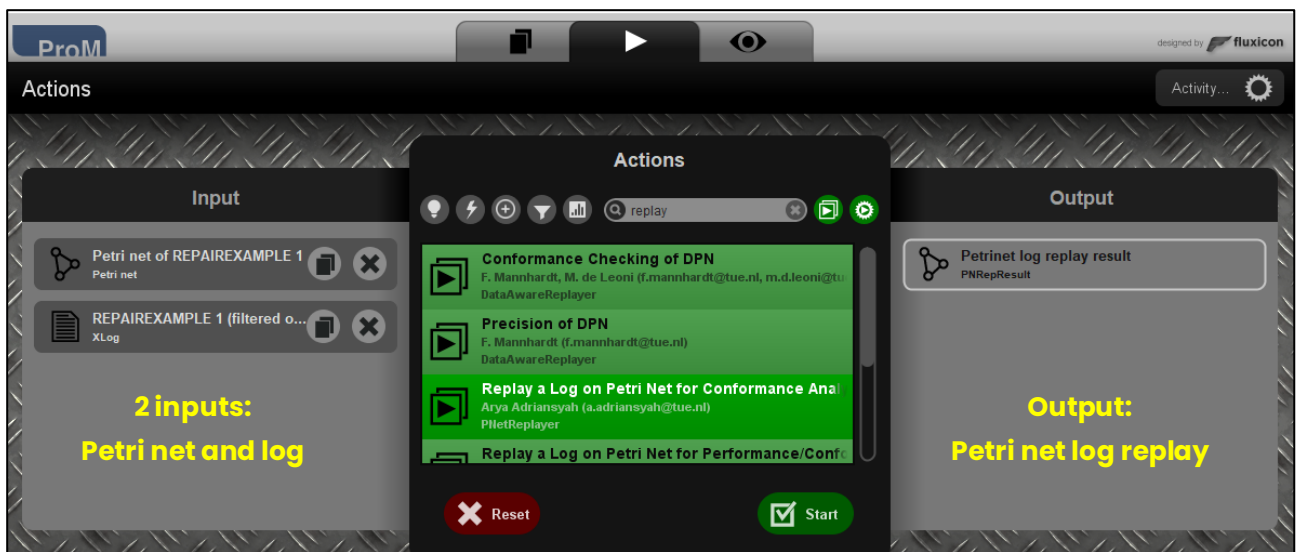


Figure 5. Replay a Log on Petri Net for Conformance Analysis

When the plug-in is launched, the first screenshot is as in Figure 5. It aims to show the mapping between the transitions in the model represented by Petri net and event classes in the logs. An event class characterizes a set of events that have characteristics in common. Event logs define classifiers, a set of attributes, to determine which characteristics to consider. In Figure 6, we use the *MXML Legacy Classifier* that defines the pair "*concept:name*", which stores the activity name, and "*lifecycle:transition*", which identifies whether an event refers to the starting or the completion of an activity.

The event classifiers matched are suitable for our conformance checking; thus, we can click on the *Finish* button.

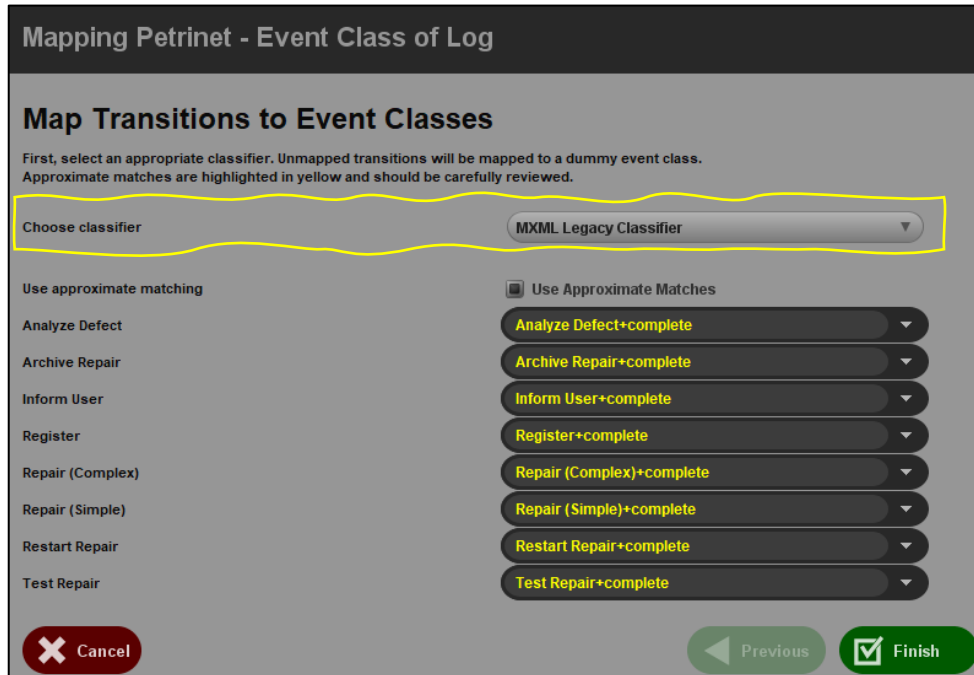


Figure 6. Mapping Petri net and Event Classes of Log

Figure 7 shows the configuration of specific parameters to build alignments. The option “*Would you penalize improper completion?*” is used to set whether or not to penalize for traces for which the tail is missing. The default is *Yes* and indicates that the process projection of the alignments must be a complete process trace that leads from the initial to the final marking. If the parameter is set to *No*, the process projection of the resulting alignments needs to be a process that starts from the initial marking but does not necessarily ends at the final marking (i.e., it is a beginning of a complete process trace). The other parameters are out of the scope of this laboratory session.

5

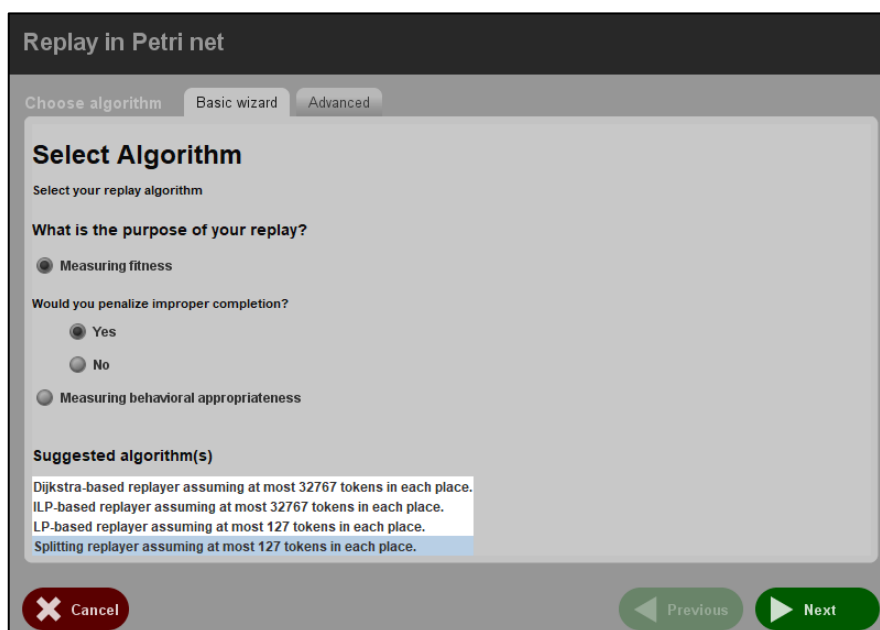
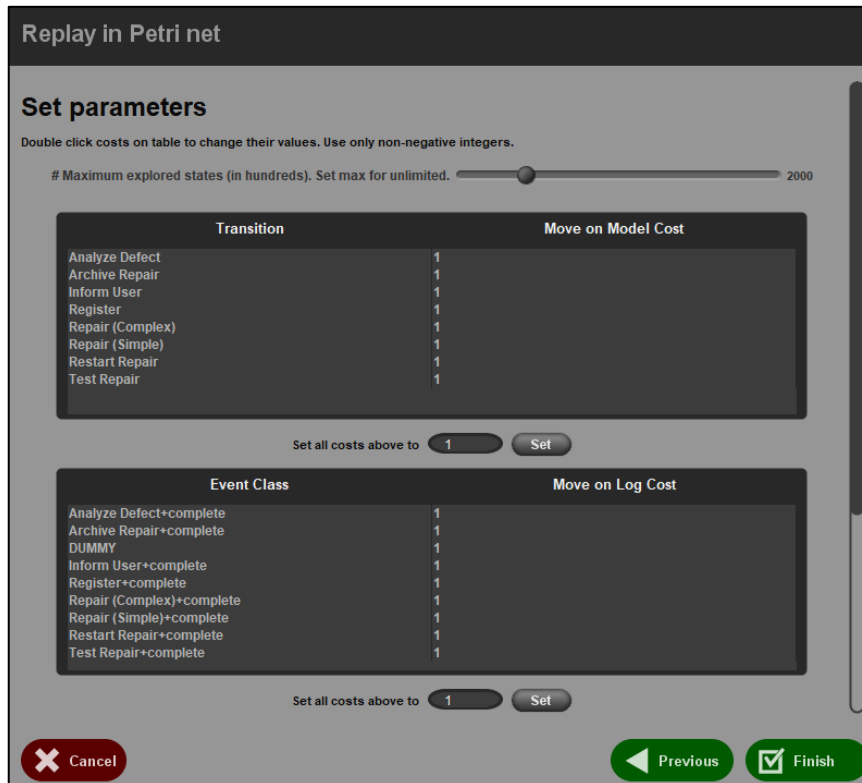


Figure 7. Replay in Petri net-1

Click the [Next](#) button without changing any parameter for the second configuration setting that allows analysts to define the costs of the alignment moves (see [Figure 8](#)). The default configuration is that every move in the model and log is given a cost equal to 1, whereas moves in both (a.k.a. synchronous moves, not seen on [Figure 8](#)) are associated with a 0 cost. We can now press [Finish](#) button to build the alignments.



**Replay in Petri net**

**Set parameters**

Double click costs on table to change their values. Use only non-negative integers.

# Maximum explored states (in hundreds). Set max for unlimited.  2000

Transition	Move on Model Cost
Analyze Defect	1
Archive Repair	1
Inform User	1
Register	1
Repair (Complex)	1
Repair (Simple)	1
Restart Repair	1
Test Repair	1

Set all costs above to

Event Class	Move on Log Cost
Analyze Defect+complete	1
Archive Repair+complete	1
DUMMY	1
Inform User+complete	1
Register+complete	1
Repair (Complex)+complete	1
Repair (Simple)+complete	1
Restart Repair+complete	1
Test Repair+complete	1

Set all costs above to

Figure 8. Replay in Petri net-2

The moves of all alignments are projected onto the Petri-net model. It allows us to quickly gain an insight into frequencies of occurrences of activities and deviations. Each transition is filled in with a blue colour whose intensity increases with the number of moves for that transition. Inside each transition, two numbers below the activity name are shown in brackets  $(x/y)$ , indicating the alignments contain  $x$  synchronous (occurred both in the log and model) and  $y$  only model moves for the transition. Considering that the event log contains incomplete traces, it is thus not surprising that [Archive Repair](#) includes 50 model moves. It means an event for the last activity is missing in 50 traces. As we discussed, it is missing because the event log was extracted when those 50 traces were still being executed.

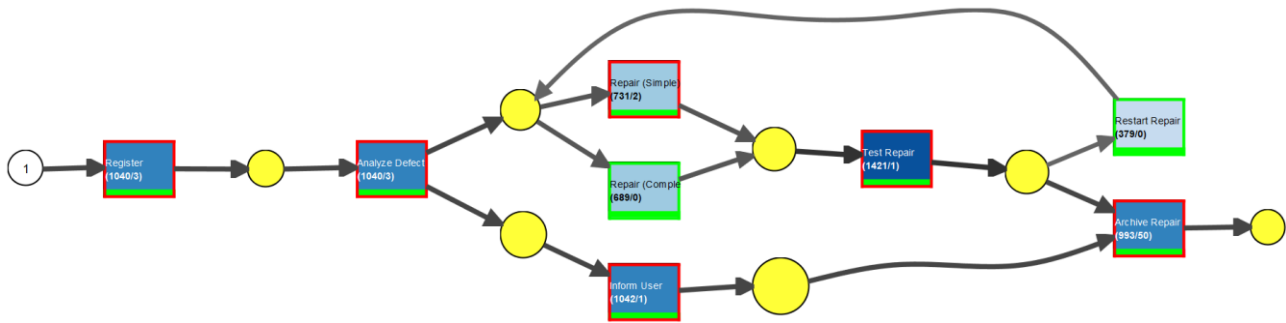



Figure 9. Alignment result for the model in Figure 3 and repair's event log

### B. Experience on a Second Event Log related to the Repair Process

The repair's first event log is largely compliant with the model in Figure 3, which is unsurprising given that the model was discovered from this first event log. The scenario shown in Figure 10 is undoubtedly more interesting. Alignments (and fitness) are computed using a second event.

The second event log is presented in the zip file under repairExample2.xes. The start events are already filtered out, and alignments are computed with respect to the model in Figure 3.

Figure 11 shows the projection of the alignments onto the model. We have already discussed how the color darkness reflects the frequency of moves for each transition. A bar (  ) is shown at the bottom of each transition, which is largely green in this example. It shows the ratio between synchronous moves and model moves for the activity. The colours reflects the share of synchronous moves (green) and model moves (purple), respectively. The bars are predominantly green, which implies that most of the activities are frequently occurred in both the model and log. In other words, they are rarely involved in model moves. The activity *Repair (Complex)* never happened in the only model move with the transition numbers of 694/0.

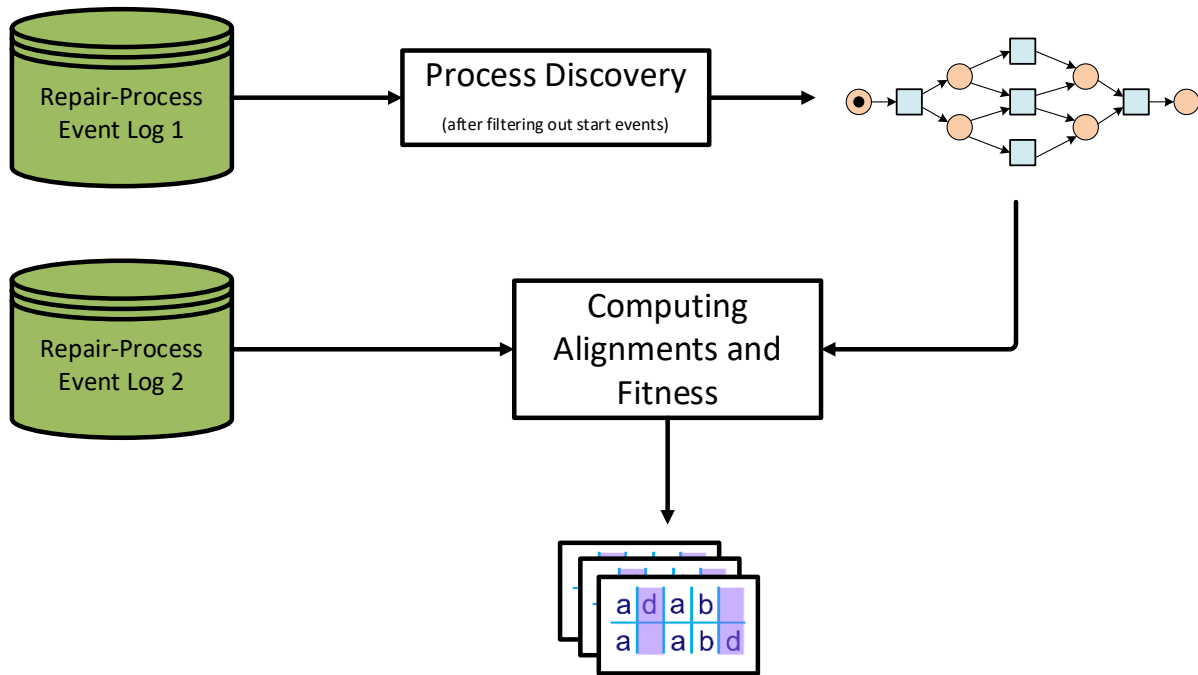


Figure 10. Schematic presentation of the second event log use

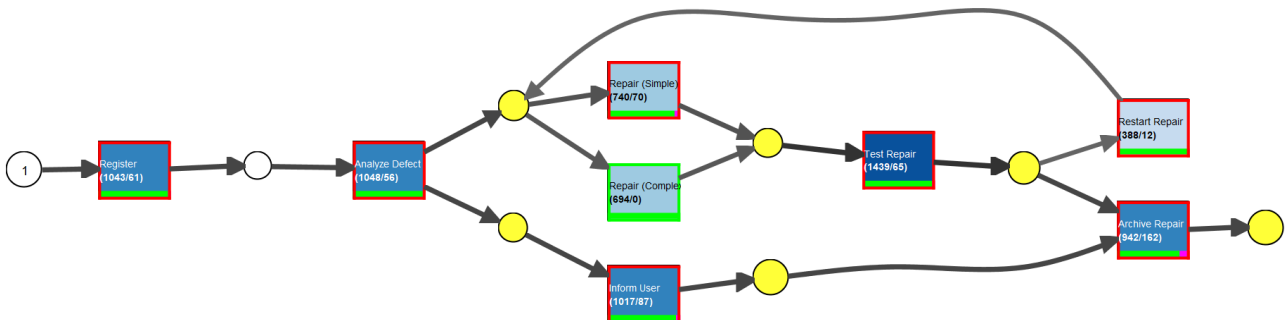


Figure 11. Alignment result for the model in Figure 3 and repair's second event log

The proportion of the **purple bar** is slightly larger for *Archive Repair*. Observing the information inside the transition, *Archive Repair* occurred 942 times in the synchronous and 162 times in the model. More information can be obtained by clicking on the activity *Archive Repair* and expanding the drop-down option *Element Statistics* under the *Info tab* in the *Inspector* (pop-up in front of the screen).

Some places are filled in yellow. These are places that contained tokens when moves in the log were observed. The size of the places indicates the frequency of moves in the log. From the analysis of [Figure 11](#), we can see several moves in the log when the transition *Inform User* is enabled because it is the only place with a larger and yellow-colored.

In general, plug-ins may provide different types of visualizations for the output. One can change the visualization by clicking on the combo box on the top right-hand side. For the alignment-based conformance checking, there are several alternative visualizations. Let's focus on the additional visualization called *Project Alignment to Log (PNetReplayer)*.



The invocation of *Project Alignment to Log (PNetReplayer)* plug-in leads to the visualization in Figure 12 for this model and log. The screen is divided into two parts. Note how the left-hand side part is scrolled down in the figure.

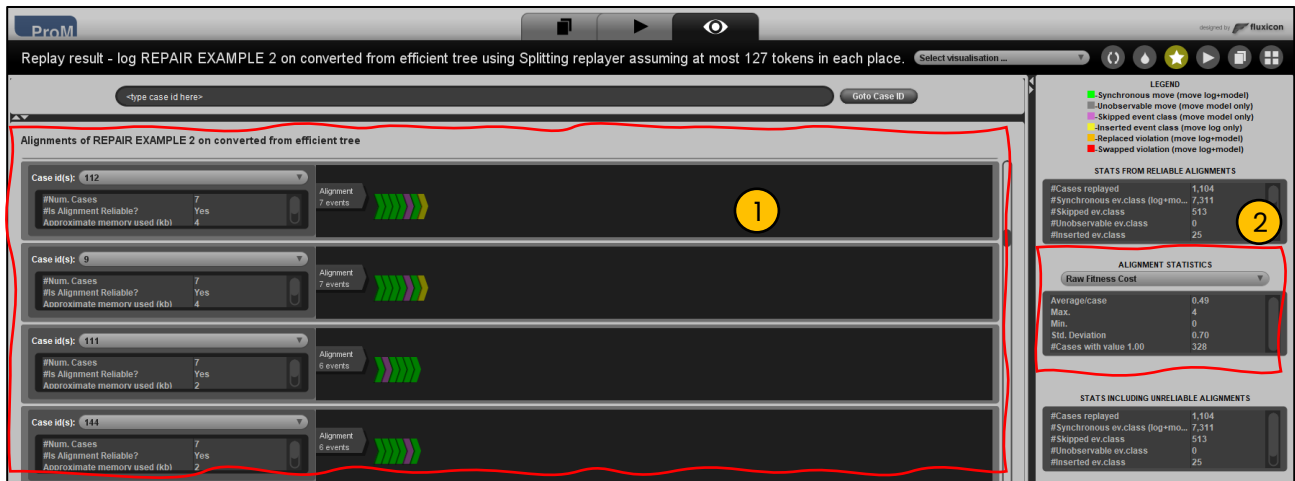


Figure 12. Project Alignment to Log (PNetReplayer) visualization

In Area ① shows the alignments of single traces. Traces that are sequences of the same events are grouped together since they yield the same alignment. Each alignment is visualized as a sequence of triangles where each triangle is a move. Each triangle is filled with a colour that depends on the type of moves: green triangles represent synchronous moves; purple triangles are for moves model only; yellow indicate moves log only. The grey colour is used for moves in the model for invisible transitions. Please note that moves in the model for invisible transitions are not linked to deviations. Invisible transitions are never recorded in the event log by definition; hence, they are always involved in moves in the model.

Alignments are sorted by descending fitness. On the left of each alignment, there is a box with some information, such as the identifiers of the traces (case ids) associated with that alignment and the raw cost (the sum of the costs of the alignment moves). Also, trace fitness is shown there (ignore move-model fitness and move-log fitness). Recall that trace fitness is between 0 and 1, where 1 and 0 indicate perfect or the poorest fitness, respectively.

Area ② shows statistics over all alignments. The current configuration focuses on the alignment cost. It states the average cost of the alignments of the different traces is 0.49 with a standard deviation of 0.7, where the minimum cost is 0 (perfectly fitting), and the maximum cost is 4.

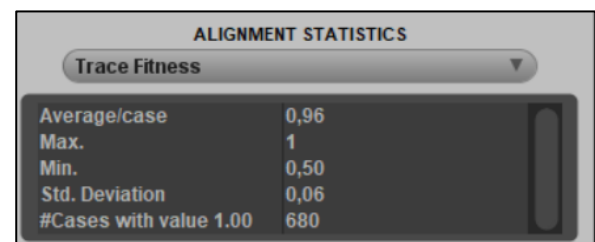


Figure 13. Alignment Statistics – Trace Fitness

To obtain statistics on the fitness, we need to change the combo box in the Alignment statistics to *Trace Fitness*, thus getting the results in [Figure 13](#). It is computed 0.96 on average. *#Cases with value 1.00* is the number of traces that are perfectly compliant with the model.

## 2. Precision and Generalization of Process Models

As mentioned, the *fitness* measures how much each event-log trace complies with the process model. However, another important criterion is ensuring that the model does not allow for too much behavior compared with the amount of behavior observed in reality. This is measured through *precision* and *generalization*.

A process model is *precise* if it allows for all and only the behavior observed in the event log representing reality. If more behavior is allowed by the model, but this is never observed in real, the model becomes less precise. However, the event log is often incomplete because there may be behavior that has not been recorded in the event log but is likely to be observed in the future if the event log would have dumped later with more traces. Therefore, the model should be precise but, at the same time, be able to generalize to account for behaviour that will likely be observed. The latter measure is named *generalization*.

The ProM plug-in named *Measure Precision/Generalization* would compute the value for precision and generalization, where the values 0 and 1 indicate a very poor or very good value for the quality metrics of precision or generalization, respectively.

Here we continue from the repair's last event log (*repairExample.xes*). The input for this plug-in is composed of the model, the event log, and the alignment results (see [Figure 14](#)). The configuration panel can follow the plug-in invocation and contains the option *Consider traces with the same sequence of activities as one trace class*. If the option is selected, precision and generalization will be computed after ignoring the repetitions of the same traces. If we leave the option selected (the default), the result is as in [Figure 15](#), indicating that the model is very precise and generalizes well.

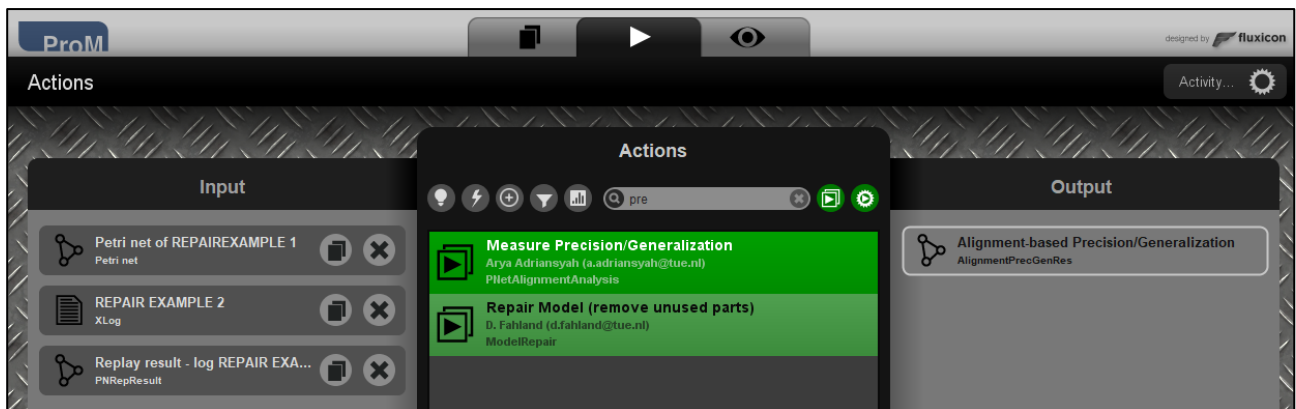


Figure 14. Measure Precision/Generalization with Petri net, log and replay result

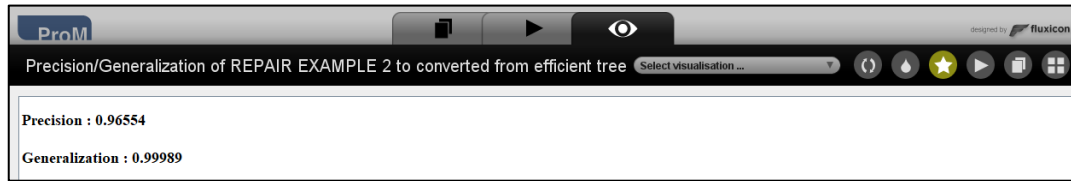


Figure 15. Precision and generalization scores

### 3. A real-life example: The road-traffic fine management process

During the laboratory session on Process Modelling via Woped, you had to fix a Petri-net model to handle road-traffic fines by Italian police. A good model is stored in the file *PetriNets-Exercise2Solution.pnml*.

The file *RoadFineLog\_30000\_traces.xes.gz* contains real-life event data of a real local police officer's actual handling. We aim to verify whether

- the model is precise and generalizes sufficiently (i.e., without under- and over-fitting)
- the actual executions comply with the national law.

It will be necessary to provide additional information to compute alignments. At first, it will be asked to specify the final marking. Specifying which places will contain one token in the final marking is necessary. Indeed, the Petri net was loaded through a file in PNML format, which does not store the final marking. The plug-in for building alignments will then ask whether a final marking is wanted as given in Figure 16. Alternatively, you can apply to *Create Final Marking* action after selecting Petri net *PetriNets-Exercise2Solution.pnml* like shown in Figure 17. Note that there are two separate plug-ins, *Create Final Marking* and *Create Initial Marking*.

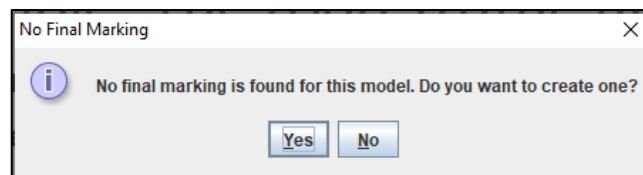


Figure 16. PNML does not have a final marking which is required for conformance checking

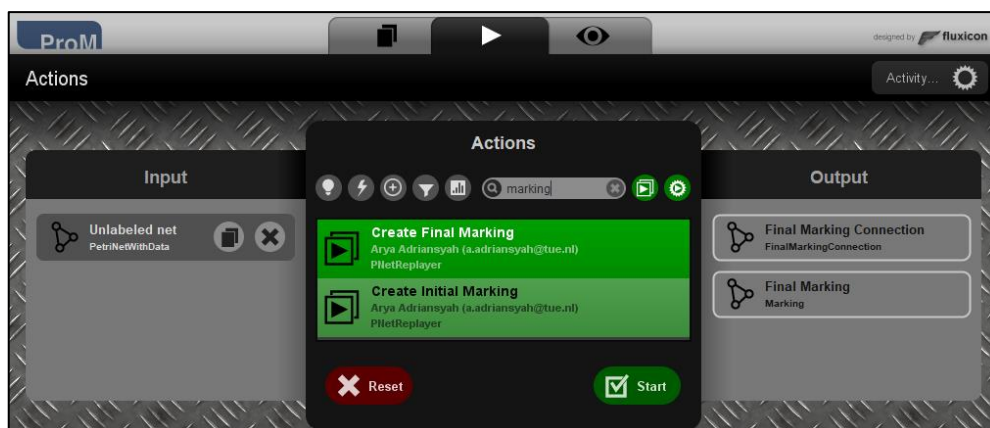


Figure 17. Creating final marking for PNML with ProM action

*Place p14* is the only part of the final marking, which is indeed the *end place* of the workflow net. Figure 18 depicts how to specify the final marking. It is necessary to select *Place p14*, then click *Add Place* and, finally, press *Finish*.



Figure 18. Selecting final marking

Figure 19 shows the appropriate mapping for the Petri-net transitions. Note how specific transitions are explicitly mapped to none of the event classes to indicate that they need to be treated as invisible transitions. Similarly, certain transitions are mapped to none of the event classes and certain event classes are going to be mapped to no transitions. The model omits to represent certain activities, such as infrequent, because their inclusion would unnecessarily complicate the model.

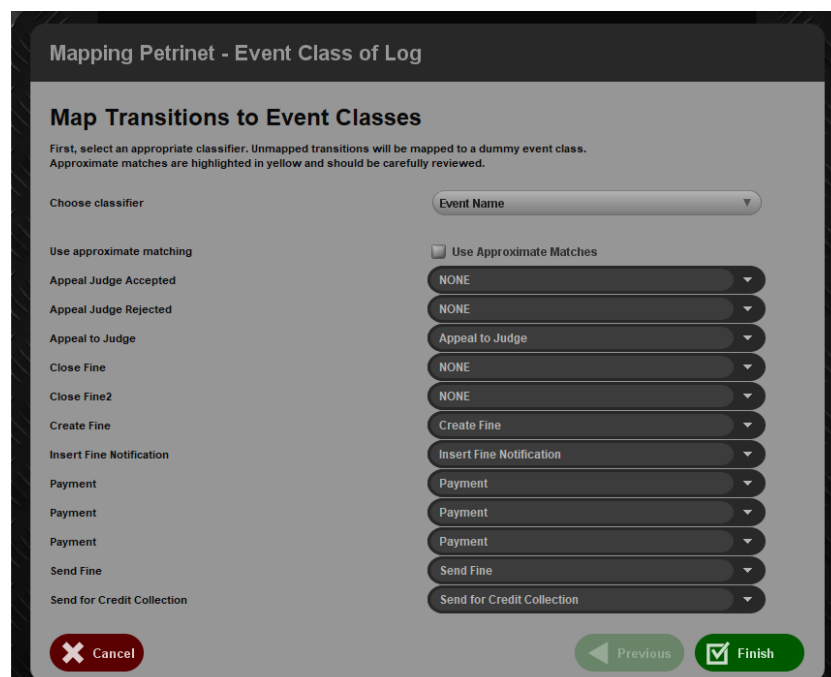


Figure 19. Mapping Petri net and event class of log for road-traffic fine management process

Figure 20 illustrates the popups that appeared because of the lack of the two mappings. Figure 20(a) informs that certain event classes are not mapped to transitions and asks to verify whether it is intended. Figure 20(b) indicates the question for unmapped transitions and asks whether those transitions should indeed be transformed as invisible. Note how important it is to state that certain transitions are invisible correctly. Invisible transitions can only be involved in model moves by definition. Indeed, invisible transitions do not leave a trail in logs through events. Thus, the model moves for invisible transitions are naturally associated with no costs, opposite to model moves for non-invisible transitions.

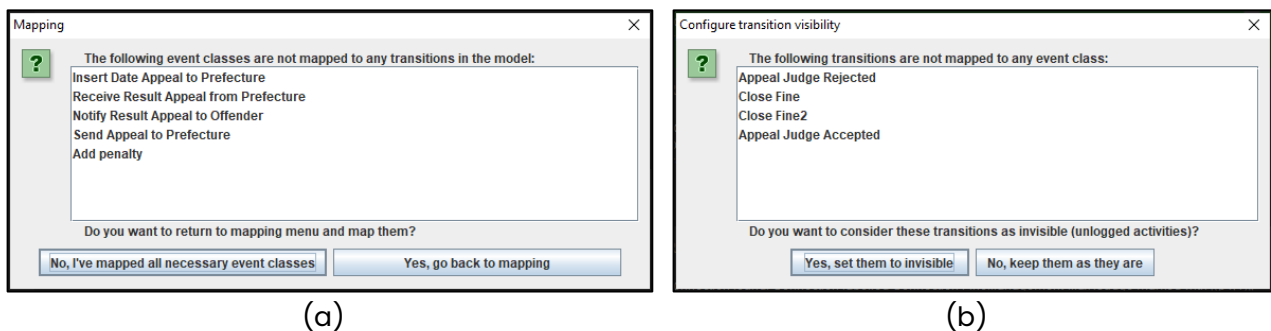


Figure 20. Additional information to provide about unmapped event classes and transitions

Figure 21 depicts the alignments of the most frequent traces. The most common deviations are related to log moves for the activity *Add Penalty*, not presented in the Petri-net model. As such, it is always involved in log moves, i.e., activities that have occurred when not supposed to.

Complete the analysis of fitness, precision, and generalization. Then, answer the following questions:

1. Is the model precise? Does it generalize sufficiently?
2. In general, what are the most common deviations? Are they frequent?
3. Does the model forget to model certain portions of behavior? If so, how can you “repair” your model to incorporate the missing parts?
4. Remove the constraint that *Appeal to Judge*. Do you expect that the precision will become higher or lower? Do you see the difference when using the plug-in?

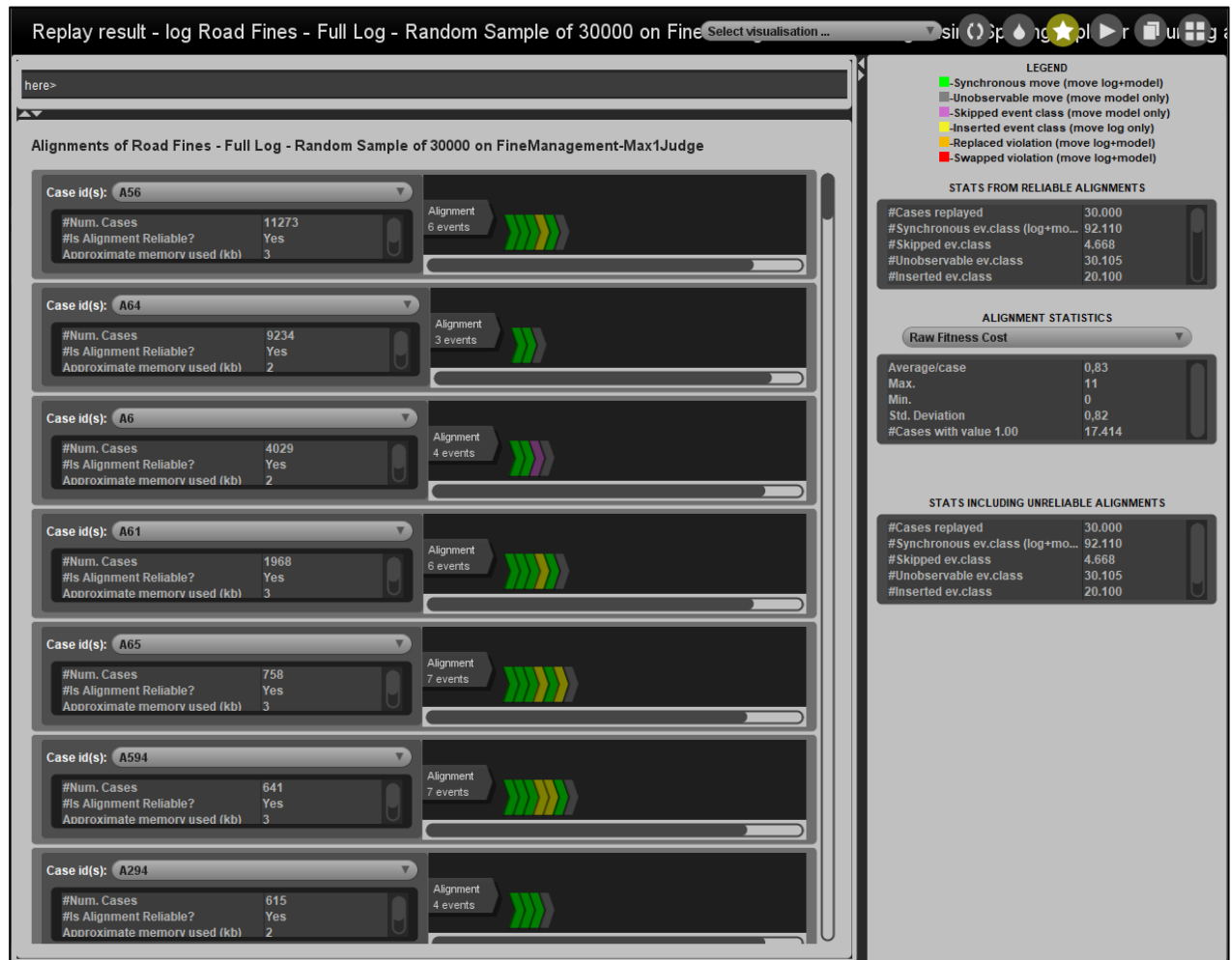


Figure 21. Alignments of the road-traffic fine management process

#### 4. Exercise

In the laboratory session on Process Discovery with ProM, you analyzed a financial process using an event log that stored executions related to 2011. In 2017, a second event log was extracted referring to executions of 2016 named *2016.xes.gz*. The systems that support the financial institute have changed. This has caused the process executions to be different.

The event logs for 2011 and 2016 store process executions, including two subprocesses: *Application and Offer subprocesses*. These two subprocesses need to be analyzed separately, as already discussed in the laboratory session on Process Discovery.

What are the executions of 2016 different from those of 2011 for what concerns the Application and Offer subprocesses?

Figure 22 depicts the application of different process mining techniques to answer the business question for the Application subprocess.

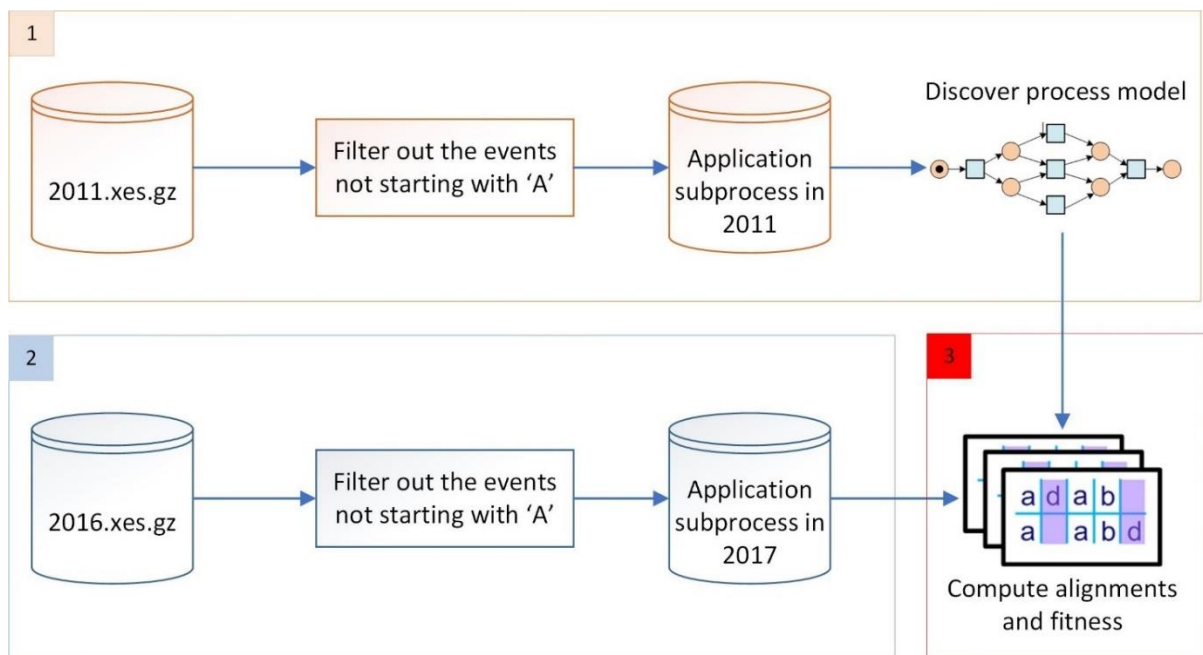


Figure 22. Schema of the Application Subprocess