Conformance Checking

▼ Overview

Conformance checking is a fundamental process mining task that involves comparing an event log with a predefined process model to identify deviations and ensure alignment between the recorded process execution and the designed model. Here are the key details and concepts involved:

Goals of Conformance Checking

1. **Fitness**: This measures how much of the observed behavior in the event log is captured by the process model. High fitness indicates that most of the recorded events can be explained by the model.
2. **Precision**: Precision checks if the process model allows for behaviors that were not observed in the event log. High precision means the model is not too permissive and only allows observed behaviors.
3. **Generalization**: This measures the model's ability to handle new, unseen cases. A model with good generalization will perform well on new data that wasn't part of the event log used for its creation.
4. **Simplicity**: This refers to the complexity of the process model. Simpler models are easier to understand and analyze, though they must still accurately represent the process.

Steps in Conformance Checking

1. **Event Log and Process Model Preparation**:
   - An event log is a collection of events recorded by an information system, detailing specific activities along with attributes such as timestamps, resources, and case IDs.
   - A process model is a graphical or mathematical representation of a process, typically created using notations such as Petri nets, BPMN, or process trees.
2. **Alignment**:
   - The event log and process model are aligned to identify corresponding activities and deviations. This involves mapping events in the log to activities in the model.
   - Alignments help in identifying where deviations occur, such as missing activities, additional activities, or activities occurring in the wrong order.
3. **Metrics Calculation**:
   - **Fitness**: Measures how well the events in the log can be replayed by the model. Techniques like token-based replay are often used, where tokens represent the flow of control in the process model.
   - **Precision**: Evaluates the model's restrictiveness. Techniques involve generating all possible behaviors from the model and comparing them with the observed behaviors.
   - **Generalization**: Assesses how well the model can handle new cases. This can involve cross-validation techniques, where the event log is split into training and test sets.
   - **Simplicity**: Quantified using measures such as the number of nodes and arcs in the model, or specific complexity metrics like cyclomatic complexity.

Tools and Techniques

- **Alignment-based techniques**: Used for both fitness and precision checking, where an optimal alignment of the event log and process model is sought.
- **Token-based replay**: Used for fitness checking, where tokens flow through the process model as events are replayed.
- **Heuristic mining**: Can be used to identify frequent patterns and behaviors that are then compared with the process model.
- **Model-based techniques**: Use formal representations like Petri nets or BPMN to define and analyze process constraints.

Benefits of Conformance Checking

- **Compliance**: Ensures that actual process executions adhere to regulatory and business rules.
- **Performance**: Identifies deviations that can indicate inefficiencies or bottlenecks in the process.
- **Improvement**: Provides insights for process redesign and optimization by highlighting discrepancies between the ideal and actual processes.
- **Transparency**: Enhances understanding and documentation of actual process behaviors compared to the designed process.

Conformance checking is a crucial aspect of process mining that helps organizations maintain control over their processes, ensuring they operate as intended and identifying areas for improvement.

▼ The Importance of Conformance Checking

A process model describes a set of possible traces, and an event log contains a multiset of observed traces (one for each case). We used this common base to compare process models and event logs. This allows us to answer questions such as:

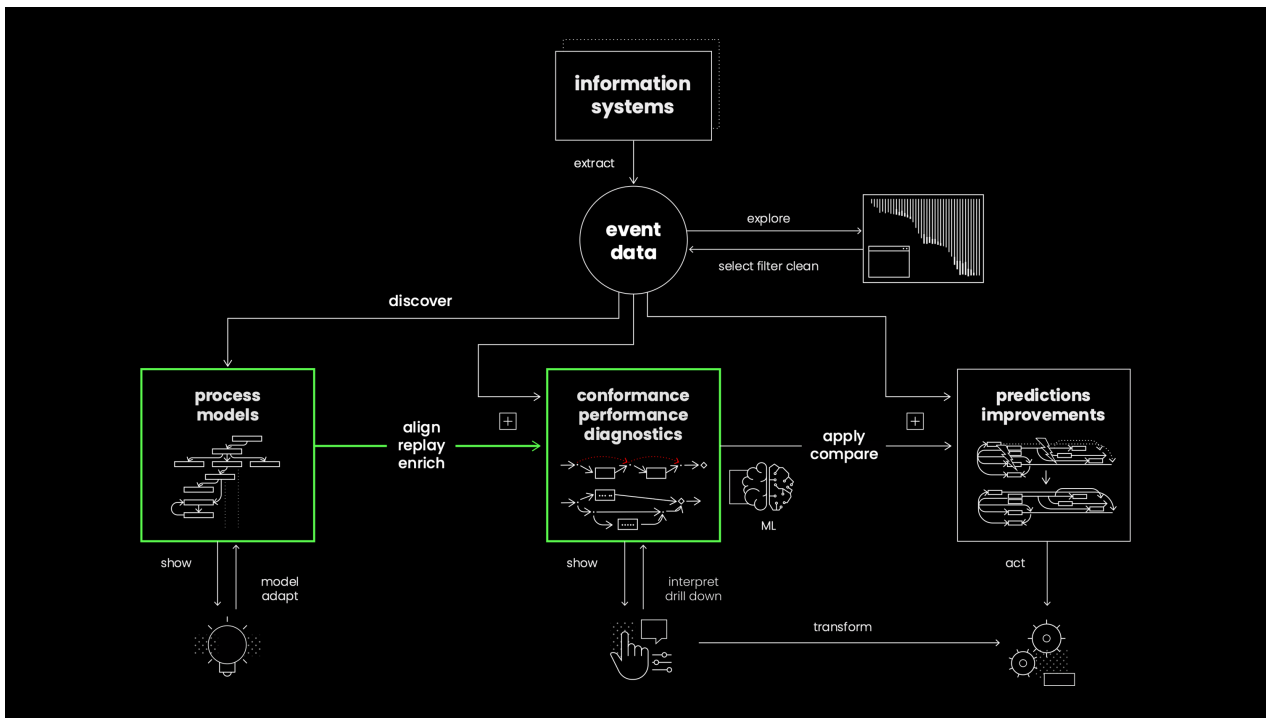
- Which traces in the log are "fitting" the model?
- What is the percentage of "fitting" traces?
- Which modeled activities are often skipped in reality?
- Which observed activities are often impossible according to the process model?
- Which (shorter) traces, possible according to the model, never happen?
- What is the precision of the model (fraction of modeled behavior actually observed)?
- What do specific deviations have in common?

There are many ways to quantify conformance. Replay fitness (also called recall) can be measured at the trace or event level. Here, we focus on fitness and control flow only. However, conformance checking has many dimensions. For example, precision measures to what degree the model is underfitting the data (i.e., the model is too general, allowing for behavior not plausible given the data). Process models can also be extended with *time constraints,* enabling conformance-checking techniques that detect deadline violations. Similarly, *resource constraints* like the 4-eyes principle and *business rules based on data* can be added. For example, the model may state that claims above 1000 euros need an extra check by the manager. These examples show that conformance checking does not need to be limited to control flow. Moreover, as we will see later, conformance-checking techniques (especially alignments) enable predictive machine-learning approaches (e.g., finding root causes for delays or deviations).

▼ Conformance Checking

▼ Concept

Process discovery starts from event data and can be seen as an *unsupervised* learning technique. This is useful for understanding the actual process and identifying unknown problems. However, in many cases, there is a *normative process mode*l describing the desired or expected process. In this case, one would like to use *conformance checking* and compare the *observed* behavior in the event log with the *modeled* behavior using a process model (e.g., a DFG, BPMN model, or Petri net) as input.



The input for conformance checking consists of an event log and a process model. The output consists of diagnostics, either quantifying the differences or annotating the event data or process model with deviations. For example, it is possible to show how often an activity that was needed according to the model was skipped. Also, the cases can be split into deviating cases and non-deviating cases. These two subsets of cases can be used to analyze root causes for non-compliance.

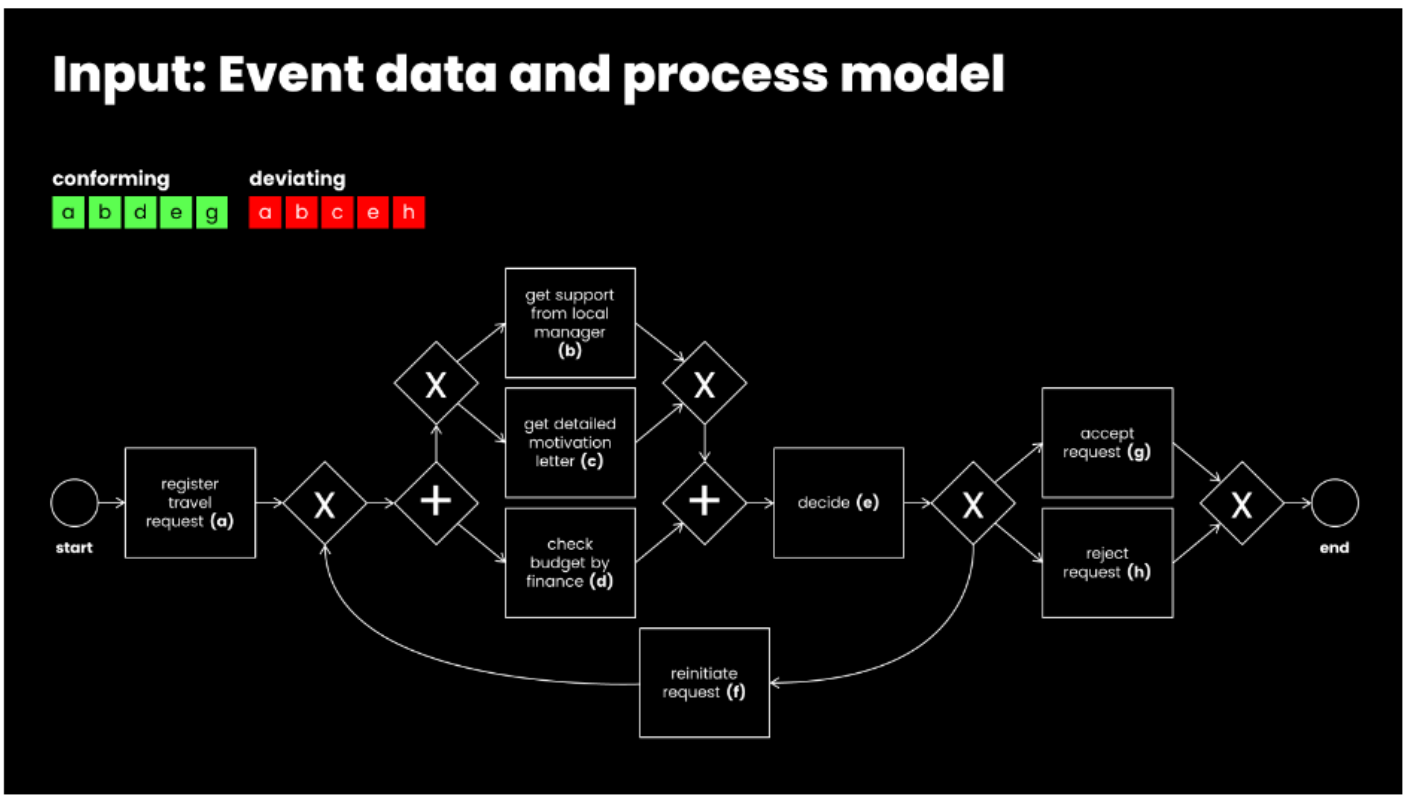
The process model can be made *manually*. In this case, it is important to connect the activities in the model to the activities in the log. It is also possible to discover the process model using process *discovery*. If we instruct the discovery algorithm to produce a model that explains all the cases in the log, then conformance checking will *not* be able to identify cases that do not fit the model. However, often we are interested in the process model describing most of the behavior seen in the log while ignoring infrequent behavior (i.e., outliers). For example, we can discover the process model based on 80% of the cases, starting with the most frequent variants. Based on such a model, conformance checking will *expose rare behavior*. This will show parts of the model that allow for exceptional behavior.

In many cases, we use a mixture of discovery and modeling. We first generate the model describing the frequent behavior and then adapt it into a normative model. This way, we can seamlessly vary between the "as is" process model describing the defacto situation and the "to be" process model describing the desired idealized process. Therefore, conformance-checking techniques can be used to check compliance and detect behavior that may be fraudulent or risky.

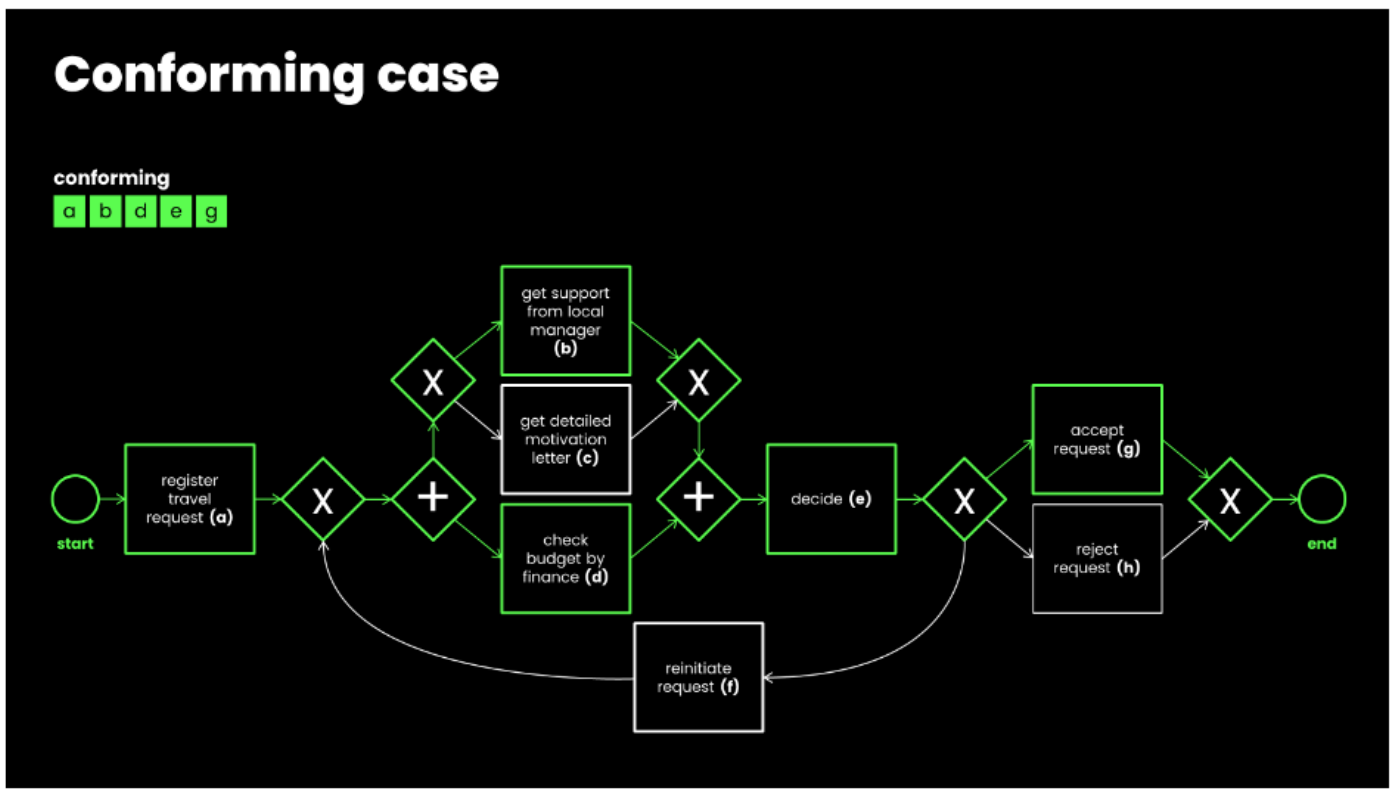
Conformance checking can also be used to evaluate a discovery algorithm. *Replay fitness* (also called *recall*) measures the fraction of cases or events explained by the process model. Of course, this is not enough to determine the quality of the discovered process. One also needs to ensure that the process model is precise enough. Hence, there are also techniques to compute *precision*. Using a combination of conformance measures, we can compare different discovery techniques or evaluate parameter settings.

For all use cases, conformance checking requires both an event log and a process model as input.

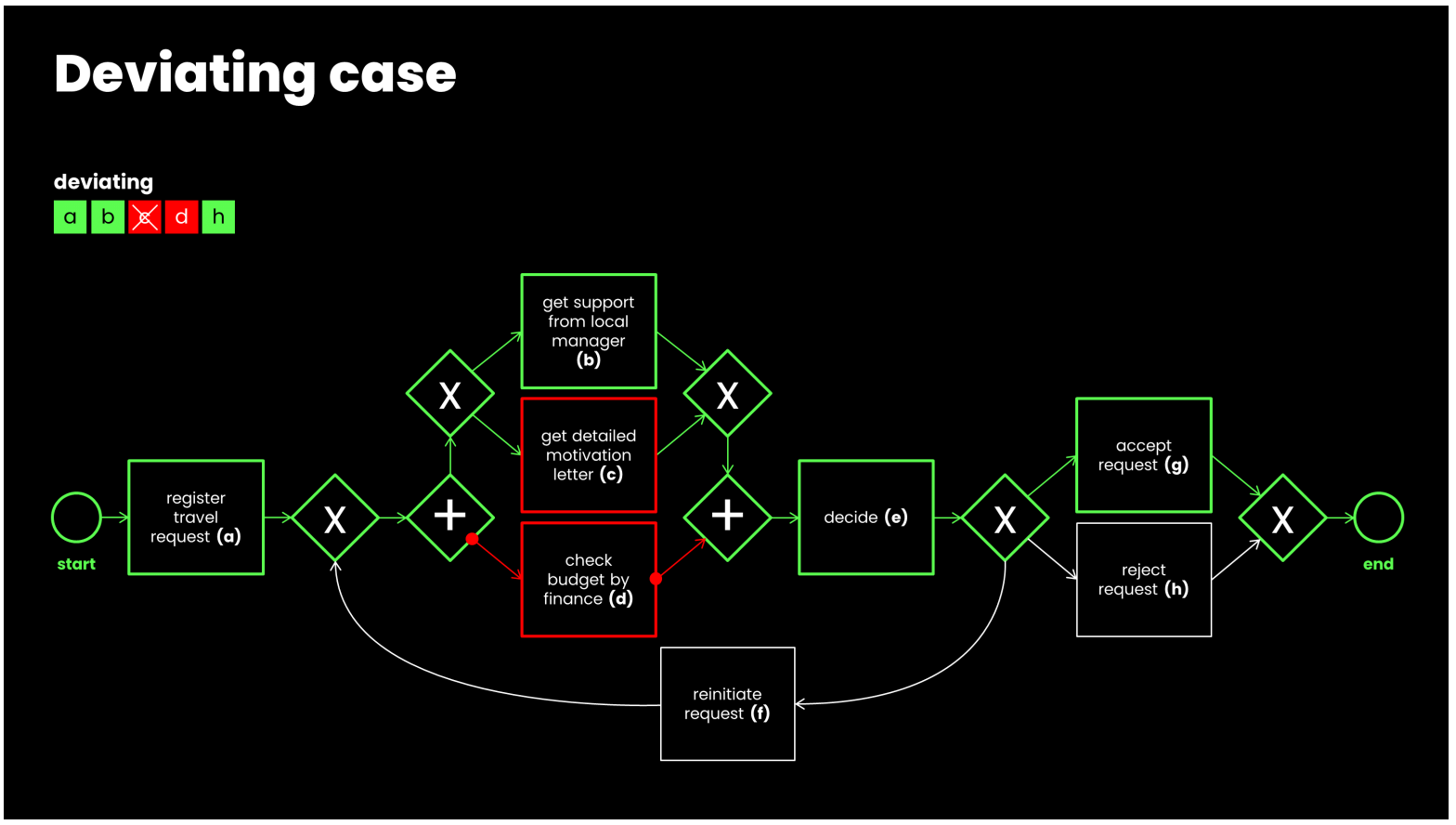
Let us take a look at a process model using the BPMN notation and two cases. Again we focus first on control-flow and describe the two cases as basic traces, i.e., sequences of activities. The green conforming case follows trace $\langle a, b, d, e, g \rangle$ and the red nonconforming follows trace $\langle a, b, c, e, h \rangle$.



Trace $\langle a, b, d, e, g \rangle$ is conforming because it is possible to replay the trace starting from the initial state of the BPMN model and ending in the desired final state of the BPMN model.



Trace $\langle a, b, c, e, h \rangle$ is non-conforming. Using conformance checking, we can see the following problems. The model indicates that there should be a choice between b and c. However, in trace $\langle a, b, c, e, h \rangle$, both b and c are performed. Activity d is concurrent to the choice between b and c, but did not happen. Hence, in reality, activity d was skipped, although this was impossible according to the model.



Focusing just on control-flow (i.e., the ordering of activities), we can identify activities that should have happened, but did not, and activities that should not have happened, but did in reality.

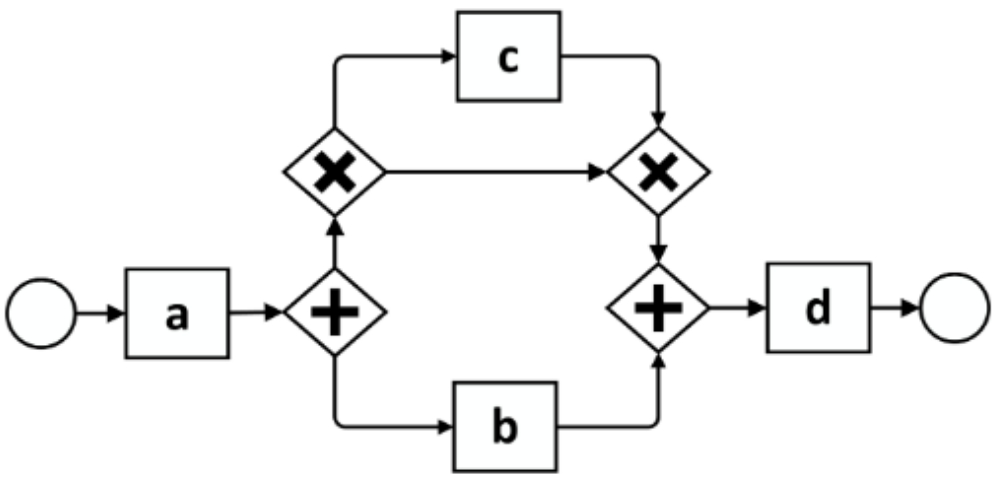
It is also possible to extend the process model and event log with other perspectives like time, resources, data, etc. For example, there are conformance-checking techniques that detect *temporal deviations* (e.g., missed deadlines). Another example would be to check the *4-eyes principle,* where two activities cannot be executed by the same resource for a given case. This is to avoid that an employee approves her own request (e.g., salary raise or business trip). Although conformance checking is not limited to control-flow, this will be our main focus.

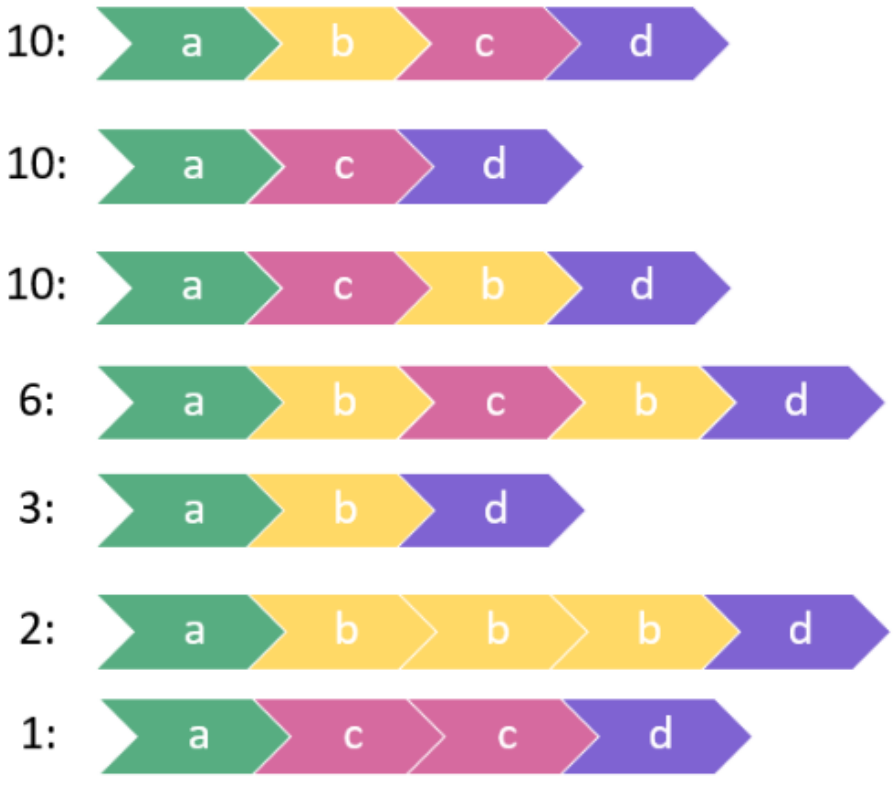
▼ Exercice

Question 1

1/1 point (ungraded)

Consider the following BPMN model and event log. How many cases are conforming to the model?





23

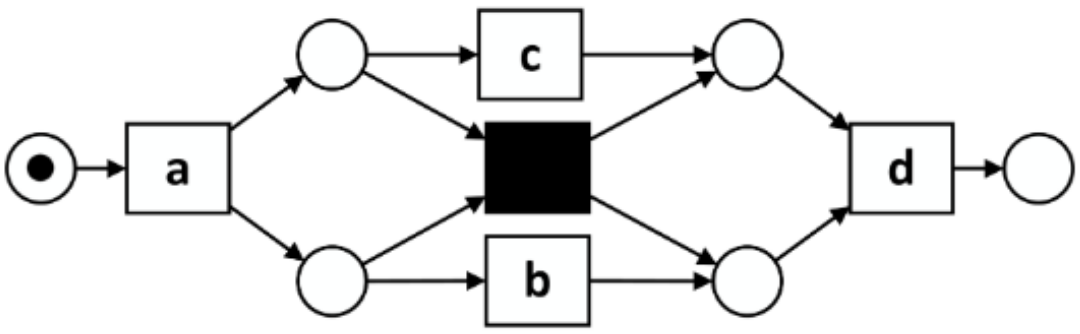
23

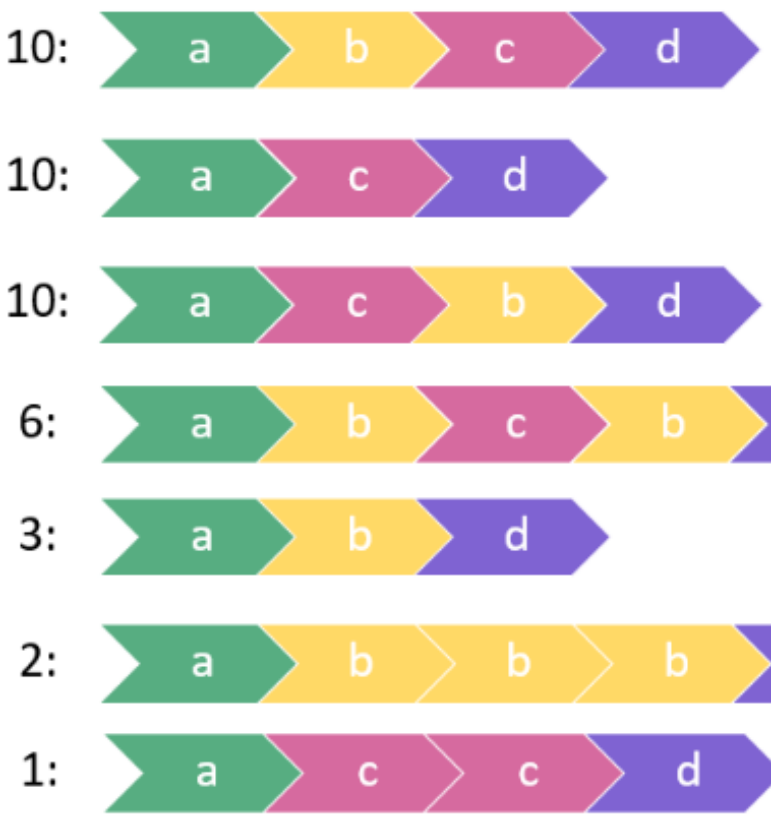
> $(a, b, c, d) : 10; (a, c, b, d) : 10; (a, b, d) : 3$

Question 2

1/1 point (ungraded)

Consider the following Petri net model and event log. How many cases are conforming to the model?





20

20

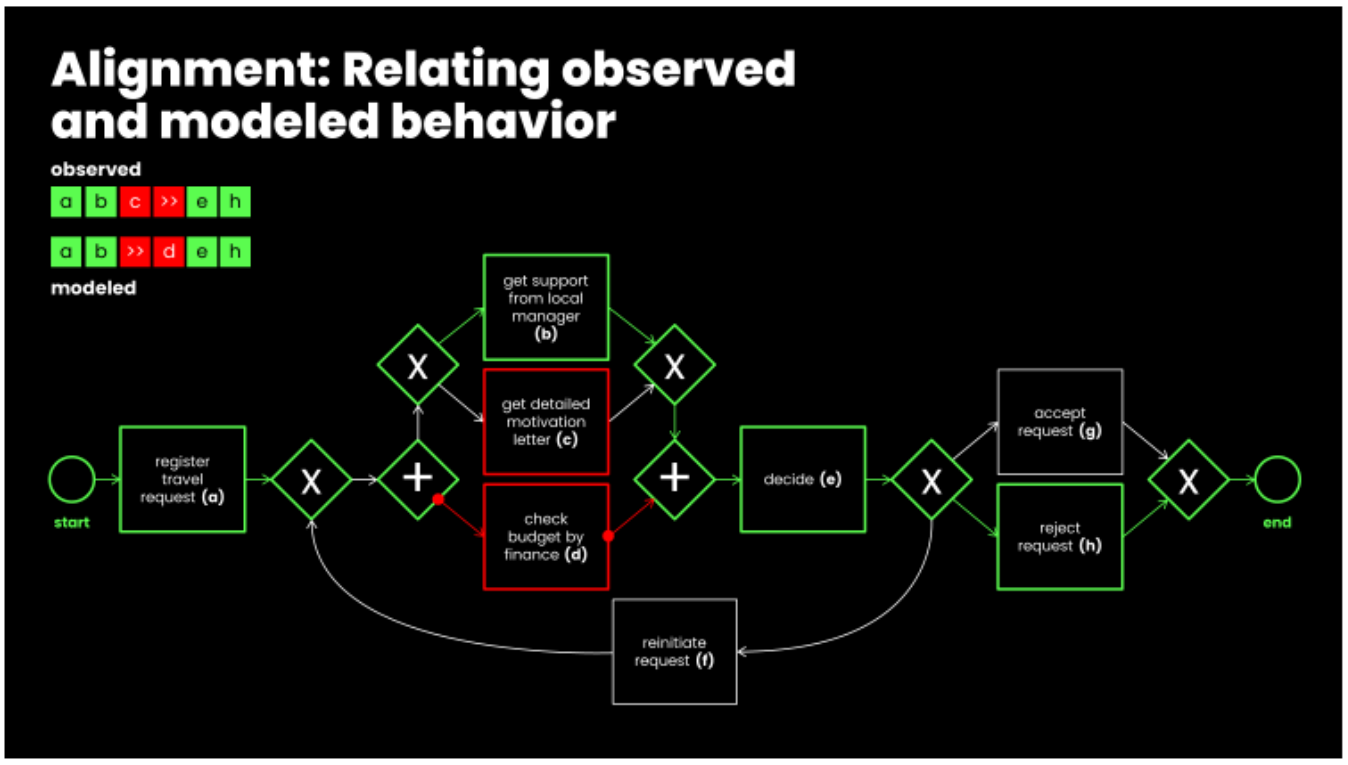
$(a, b, c, d) : 10; (a, c, b, d) : 10 = 20$

▼ **Conformance Checking Techniques**

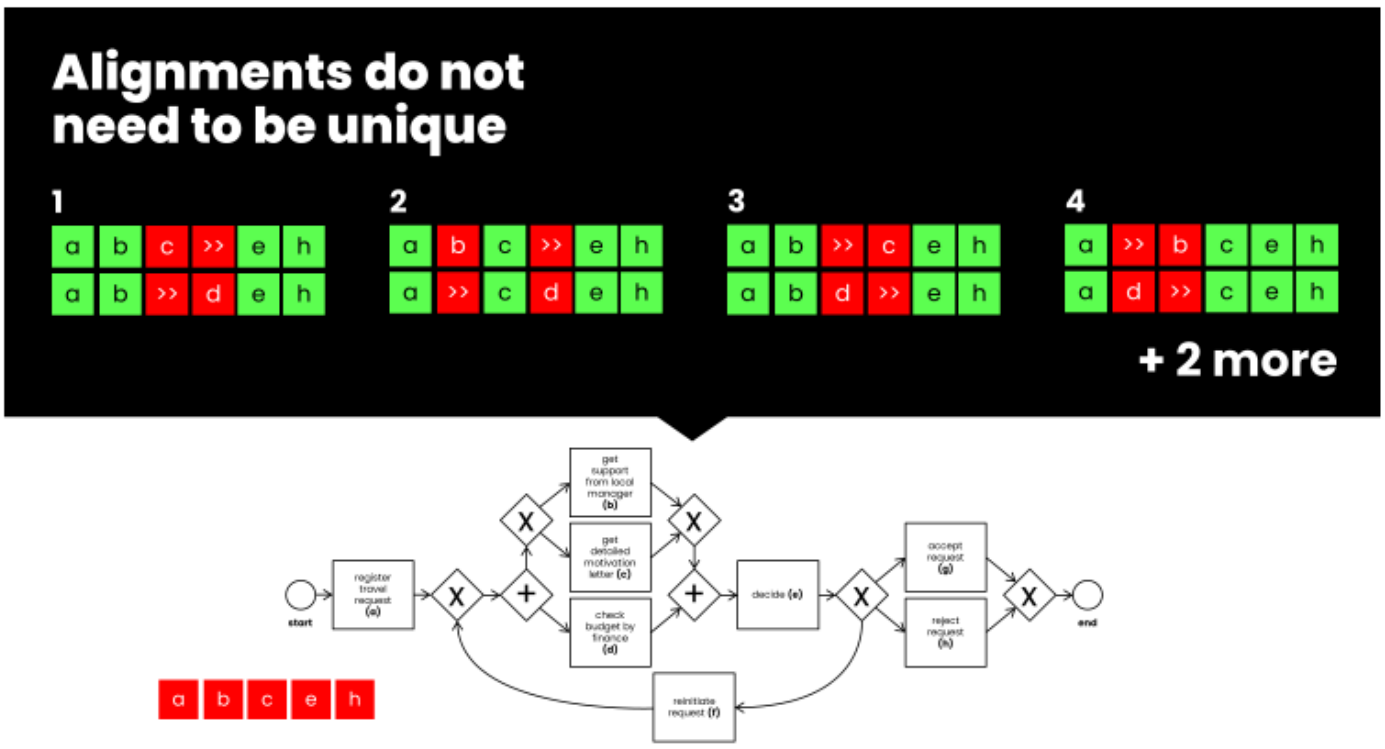
▼ **Alignment-Based Conformance Checking**

There are many different conformance-checking approaches. Here, we discuss three approaches considering the control-flow only (i.e., we abstract from time, resources, and data and only consider the ordering of activities). One of the leading approaches for conformance checking is computing *alignments*. An alignment relates events in the event log to activities happening in the process model. If there is an agreement, we say that there is a *synchronous move*. If an event in the event log cannot be matched with an activity in the process model, we say that there is a *log move* (also called "move on log"). If there is no event in the event log that can be matched with an activity necessary according to the process model, we say that there is a *model move* (also called "move on model").

Consider our running example and the deviating trace $\langle a, b, c, e, h \rangle$. We would like to find a path through the model from start to end that is as close to $\langle a, b, c, e, h \rangle$ as possible. Any such path through the model will have at least two differences: (1) one log move (because it is impossible to do both b and c) and (2) one model move (d is missing in the event log). We can denote moves by a pair of activities where the first one refers to the event log and the second to the model. For example, $(c, \gg)$ refers to a log move and $(\gg, d)$ refers to a model move. The symbol $\gg$ is called a "no move".

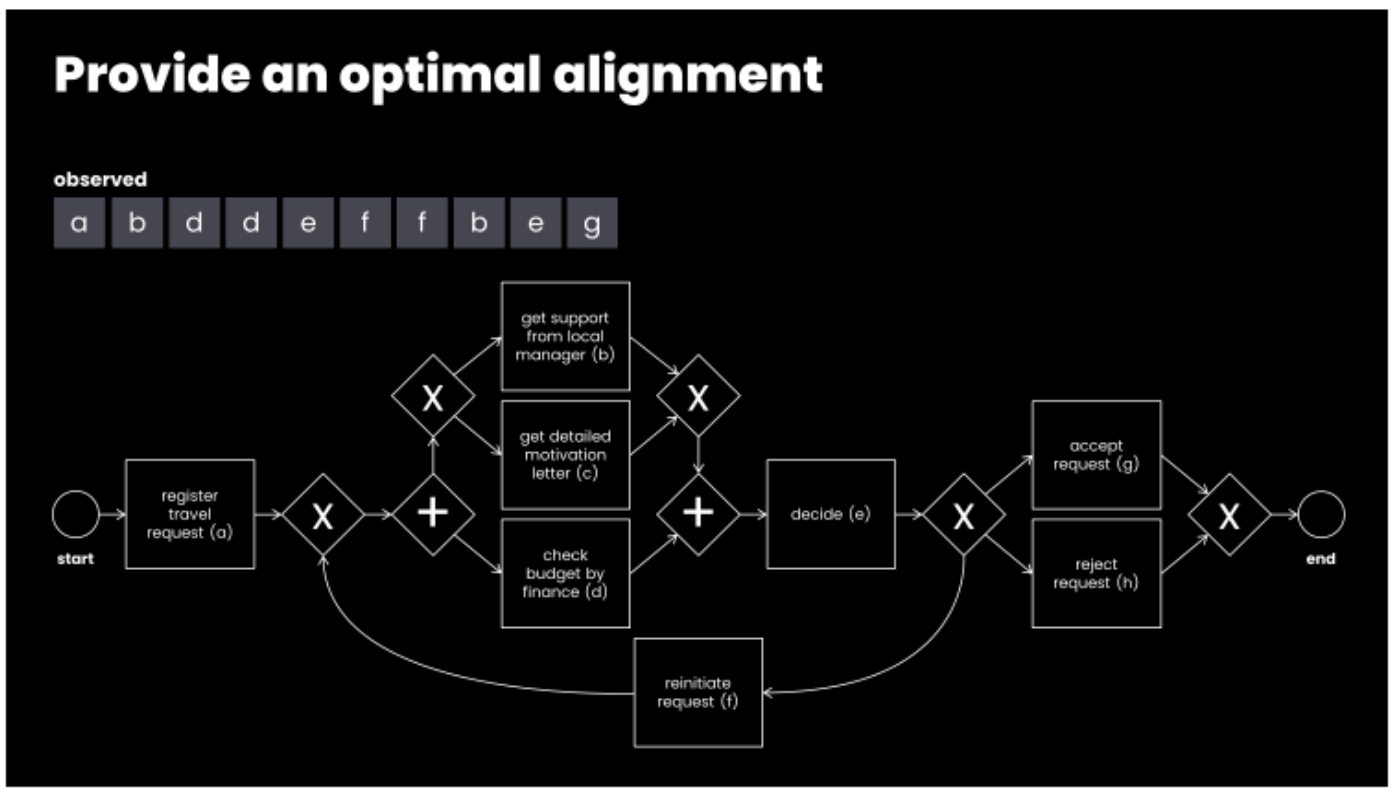


The alignment shown has four synchronous moves ((a, a), (b, b), (e, e), and (h, h)), one log move $(c, \gg)$, and one model move $(\gg, d)$. However, this is not the only optimal alignment.

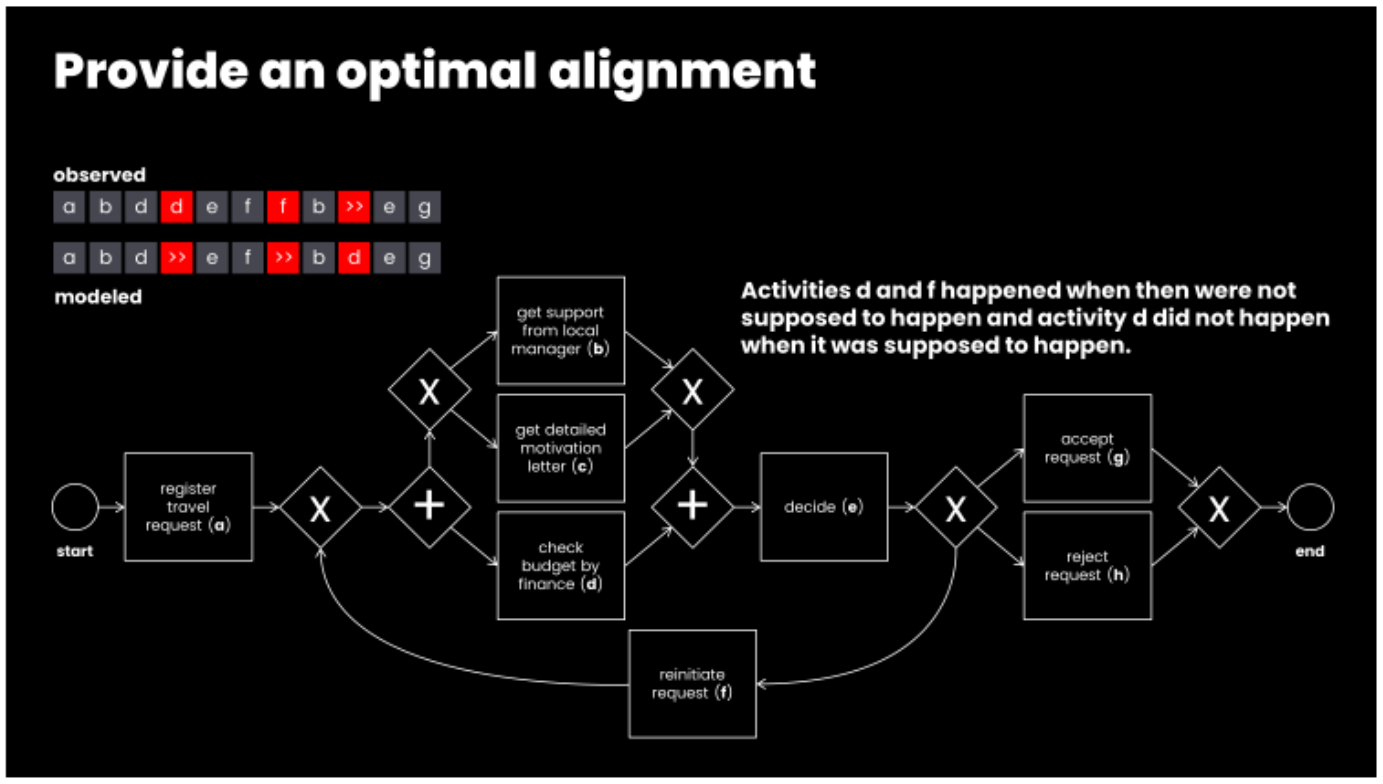


For the deviating trace $\langle a, b, c, e, h \rangle$, there are six optimal alignments. In all optimal alignments, there is one model move $(\gg, d)$ and one log model; either $(b, \gg)$ or $(c, \gg)$. The model move can be put in three different positions, explaining the six possibilities. Each optimal alignment corresponds to a possible explanation of the deviating behavior.

The example did not involve the loop in the process model. Therefore, we also consider the longer deviating trace $\langle a, b, d, d, e, f, f, b, e, g \rangle$.

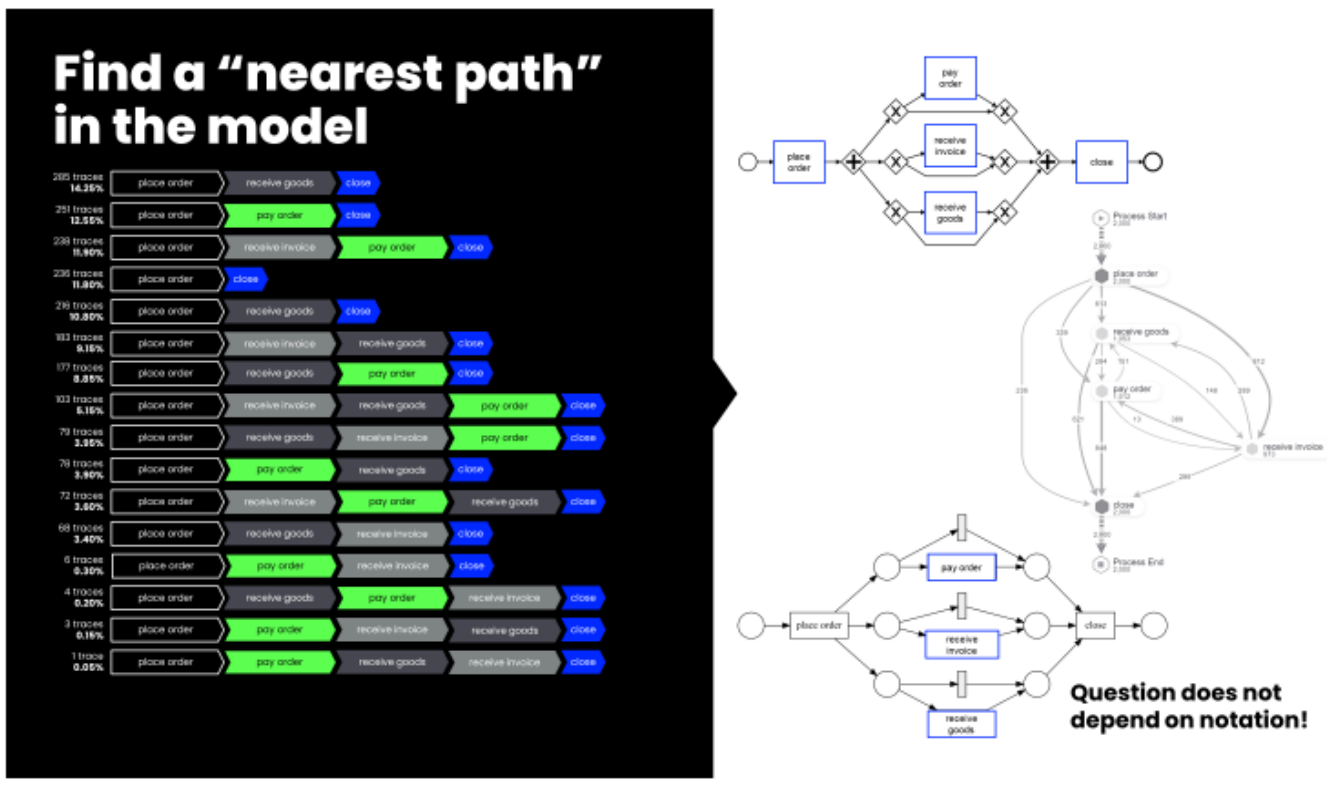


Again there are multiple optimal alignments, and one is shown below. This alignment maps the observed trace $\langle a, b, d, d, e, f, f, b, e, g \rangle$ onto the path $\langle a, b, d, e, f, b, d, e, g \rangle$ in the process model. There is one model move $(\gg, d)$ and two log moves $(d, \gg)$ and $(f, \gg)$. This suggests that activities d and f happened when they were *not* supposed to happen, and later, activity d did *not* happen when it was supposed to happen.



Note that not all nodes in the model need to be "visible" in the event log. For example, the gateways can be seen as dummy activities that are not recorded in the model. In Petri nets, we often use so-called "silent transitions" to model, for example, skipping. These correspond to "harmless" model moves, i.e., they cannot be seen in the event log and, therefore, do not correspond to actual deviations.

Alignments are not only crucial for conformance checking; they also provide the basis for *performance analysis*. Using alignments, any trace in the event log can be translated into a path through the model that is as close to the observed behavior as possible. This allows us to replay the event log on the model and analyze, for example, bottlenecks. For many advanced process mining techniques (e.g., predictions), we need to relate observed events to model elements and alignments provide the mechanism to do so.



Alignments do not rely on a specific process model notation. The notion is generic and applies to any process modeling language that describes behavior in the form of traces (DFGs, BPMN, etc.). However, if the model can be mapped onto Petri nets, there are efficient techniques to compute alignments (although for larger models and logs, one may need to resort to approximations).

The annotation *(activity, >>)* means the activity should not happen, while *(>>, activity)* means the activity should have happened.

▼ Exercice

Question 1

1/1 point (ungraded)

Select the correct statements about alignment-based conformance checking

- [x] Alignment-based conformance checking can be used as the basis to compute measures such as fitness (recall) and precision.
- [ ] Given a trace and a model, there can be only one optimal alignment.
- [ ] Since alignment-based conformance checking can be efficiently computed, it has been implemented in most commercial process mining tools.

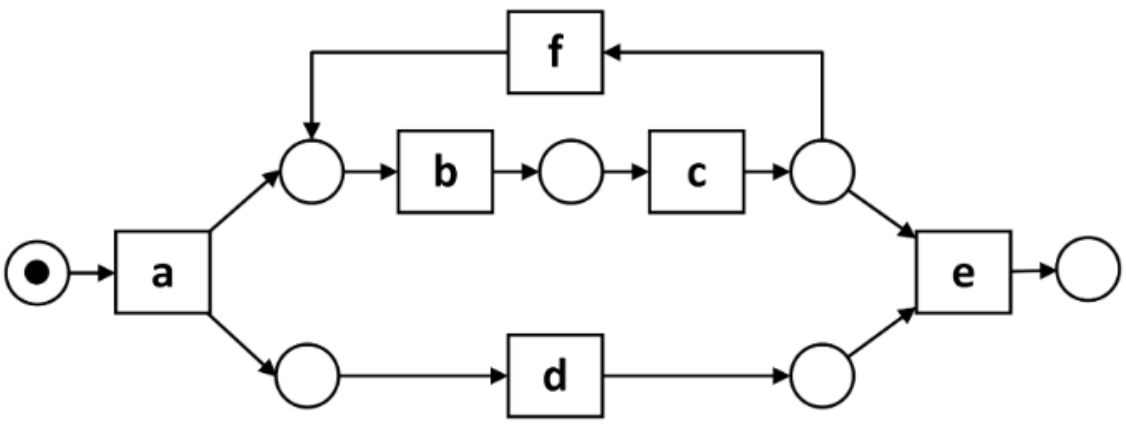
✔

Question 2

1/1 point (ungraded)

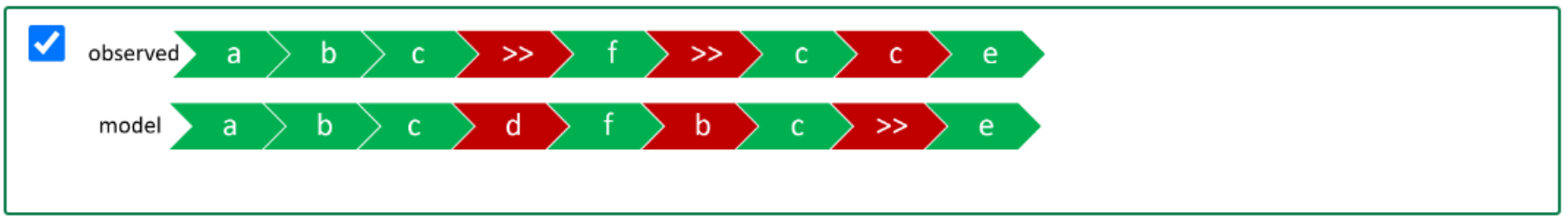
Consider the following Petri net and trace. Select the optimal alignments (the alignment(s) with the least mismatches).

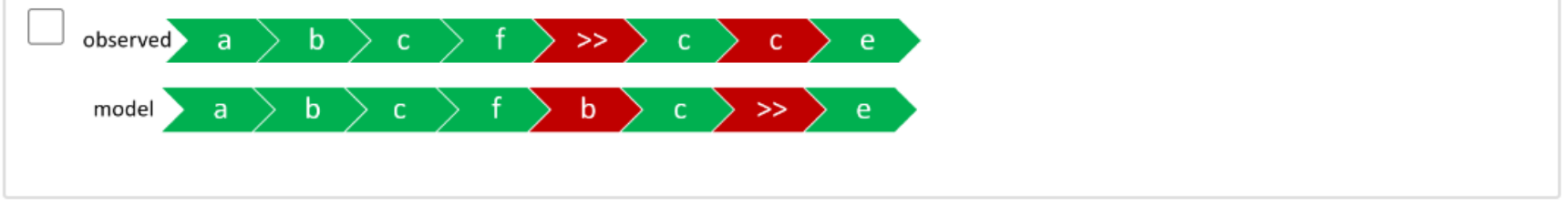
Process Model



Event log

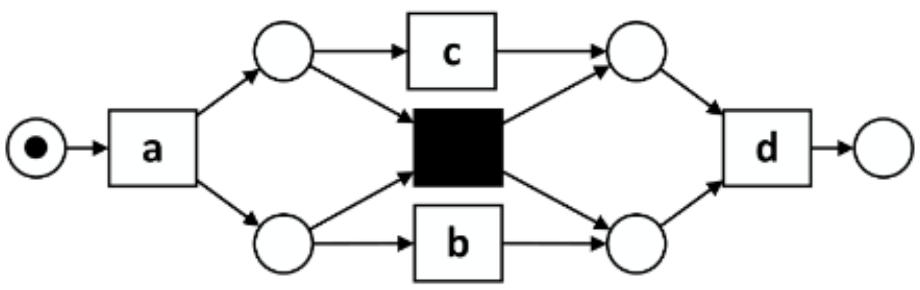
1: a b c f c c e



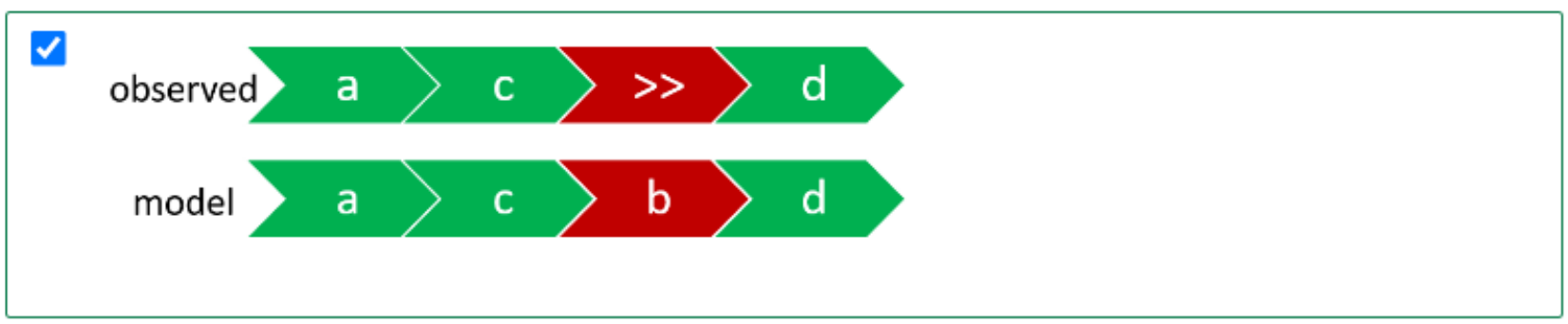


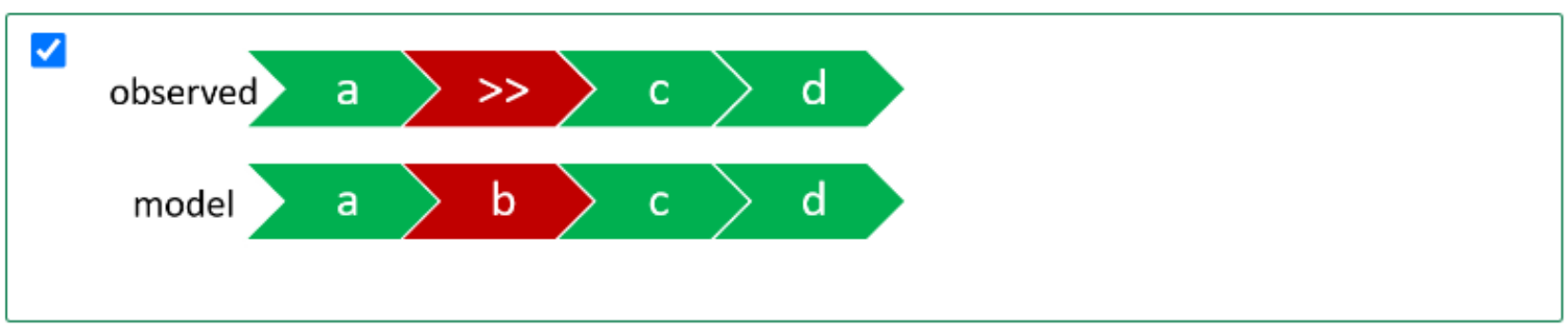
Question 3

1/1 point (ungraded)



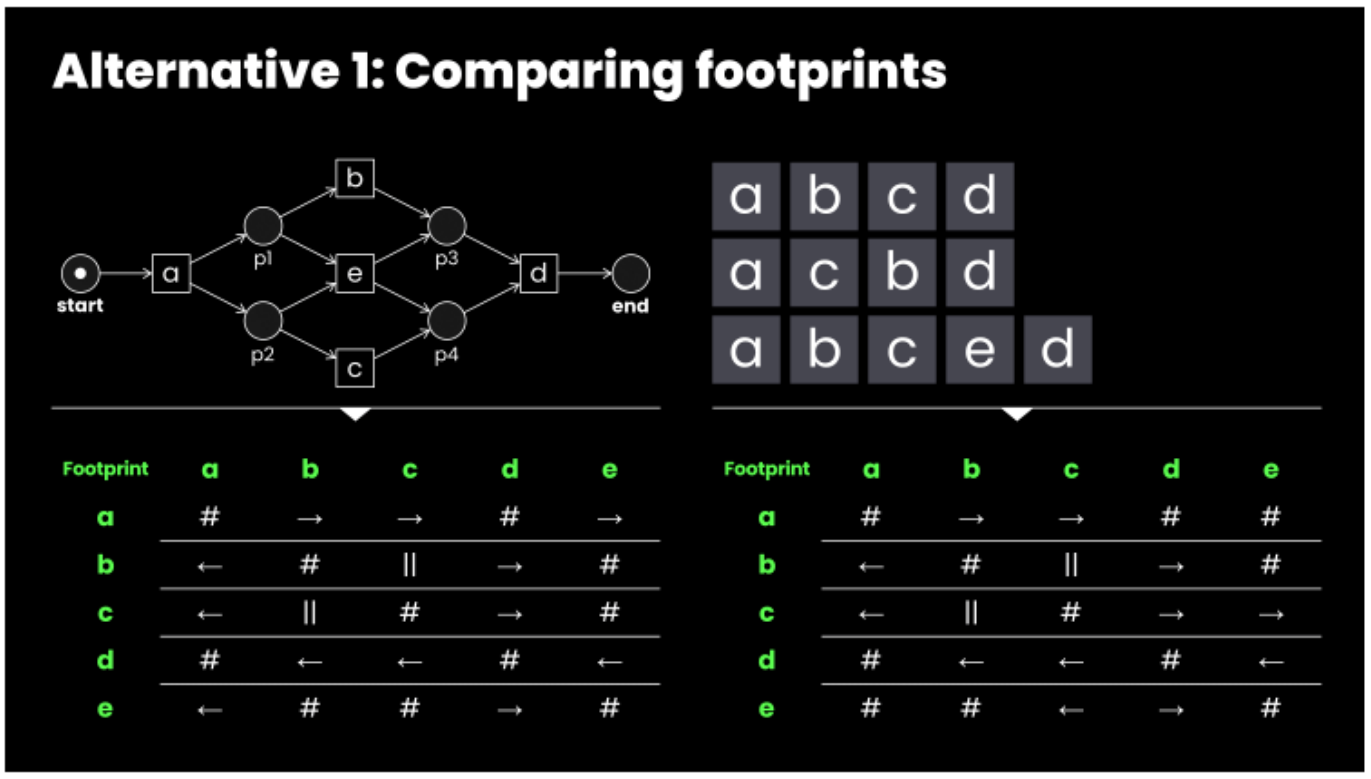
Consider the Petri net and the alignments for the trace $\langle a, c, d \rangle$. Select the optimal alignments (the alignment(s) with the least mismatches).



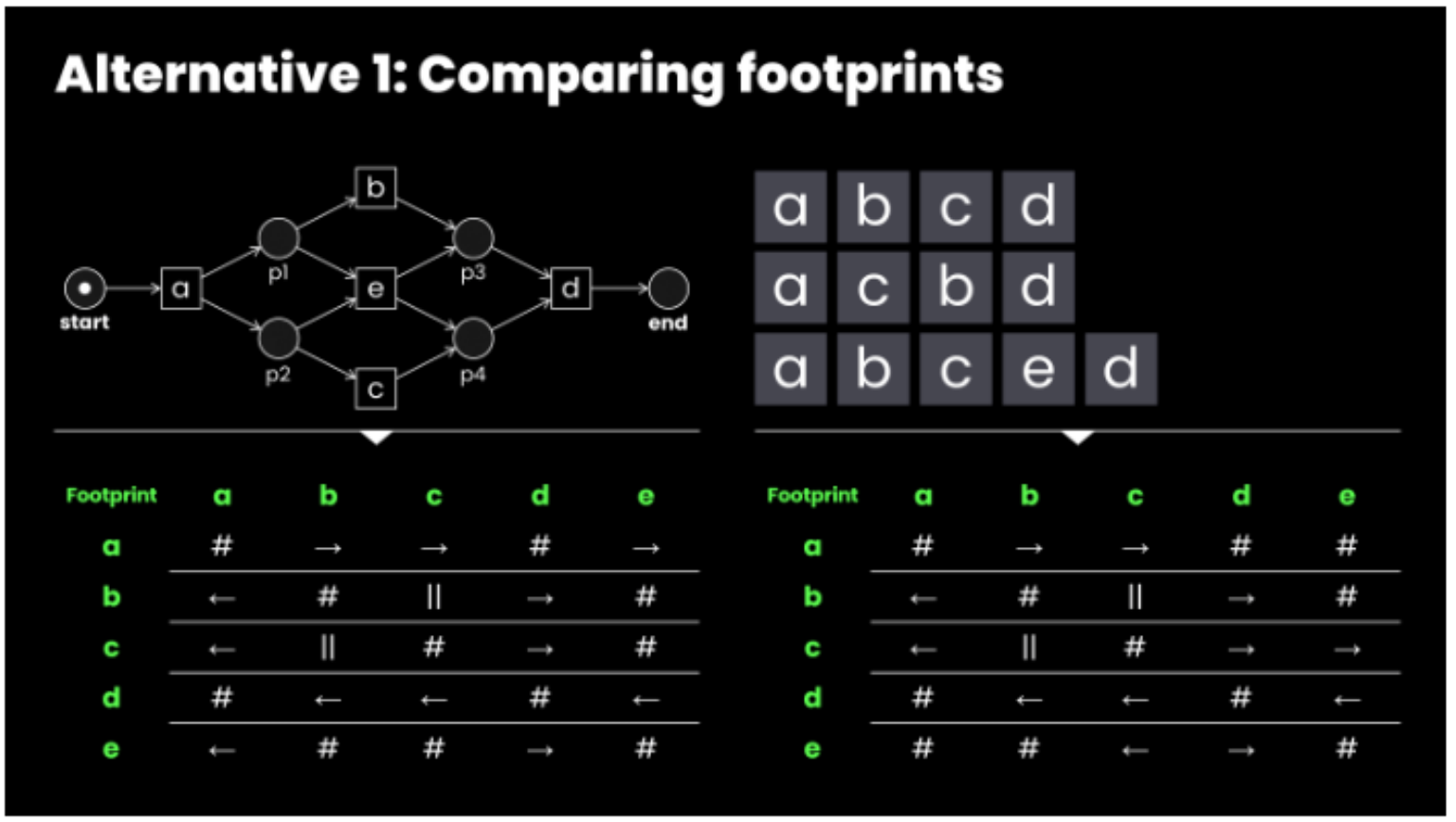


▼ Footprint-Based Conformance Checking

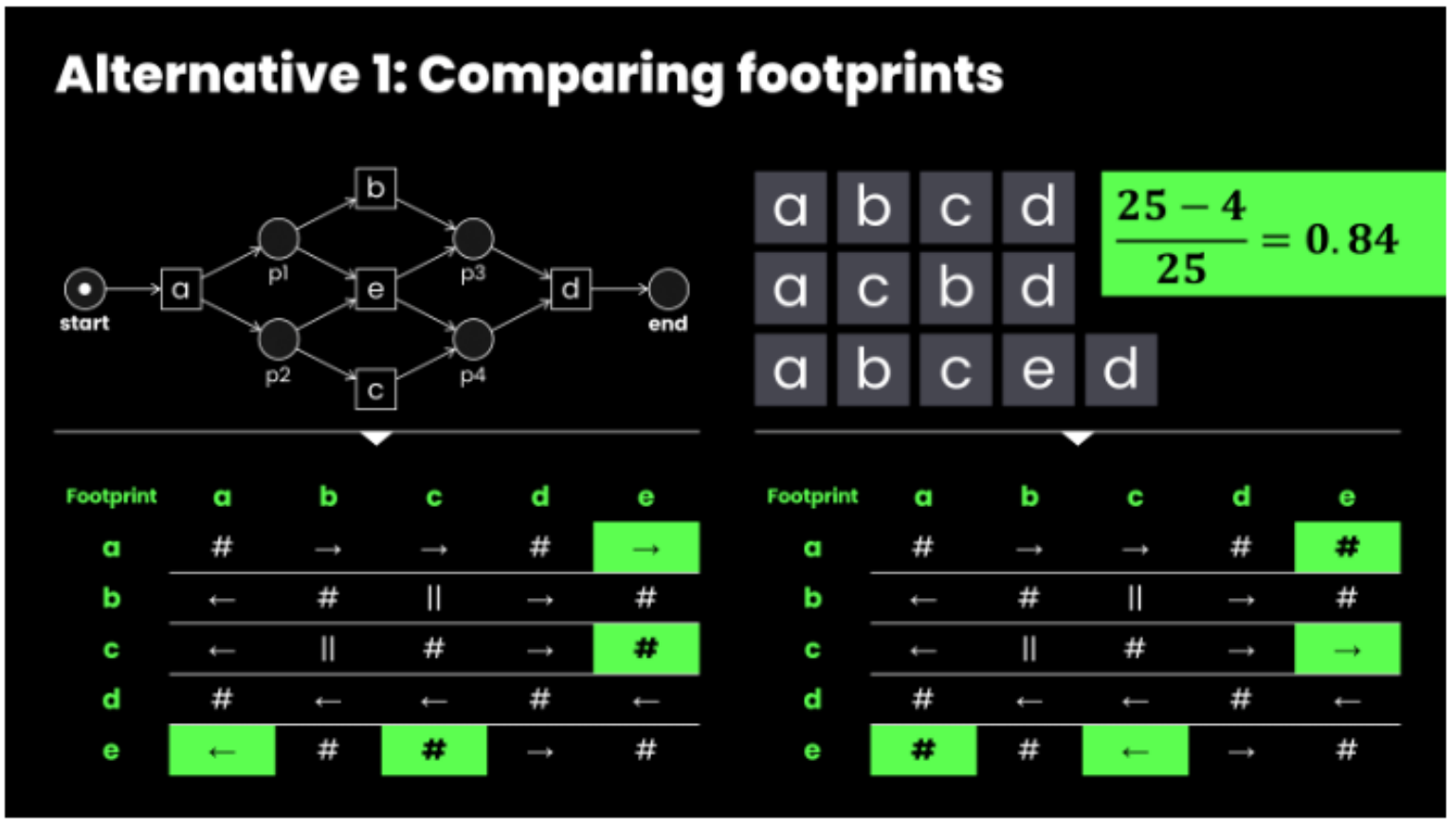
Footprints provide an alternative to alignments and can be used to compare event logs and process models using a common abstraction. The abstraction used is similar to the relations used by the well-known Alpha algorithm. For every pair of activities, one of four relations holds: $a\#b$ (a never directly follows b and b never directly follows a), $a \rightarrow b$ (a sometimes directly follows b and b never directly follows a), $a \nless b$ (a never directly follows b and b sometimes directly follows a), and $a \| b$ (a sometimes directly follows b and b sometimes directly follows a). Note that $a\#b$, $a \rightarrow b$, $a \nless b$, and $a \| b$ are based on the directly-follows relation that was also used to create the Directly-Follows Graph (DFG). Because the abstraction can be applied to event logs and process models, we can compute footprints for both.



In the above case, the model and the log agree because the footprints are identical. This is not the case when we consider the event log shown below.



By comparing the two footprints, we can compute the fraction of cells where there is an agreement.



For 21 of the 25 cells in the footprint matrix, there is an agreement. This can be turned into a value between 0 and 1. In this example, we say that conformance is 0.84 based on the footprint matrices. The cells where there is a disagreement are highlighted. For example, a can be followed by e in the model, but this never happens in the event log; also c is followed by e in the event log, but this is impossible in the model.

Note that the comparison of footprints is different from fitness. Even when all traces in the event log can be replayed on the model without any problems, the footprints may still be very different. Also, the comparison of footprints does not consider frequencies. However, it shows the idea of using a common abstraction to quantify conformance. Many other techniques use this principle, but use a more sophisticated abstraction.

A Footprint Matrix is a compact representation of the direct relationships between activities in an event log. It helps to visualize the dependencies and constraints between activities in a process. Here's how you can understand the Footprint Matrix:

Types of Relationships in a Footprint Matrix

1. **Directly Follows (→)**: Activity A is directly followed by Activity B in the event log.
    - Example: A → B means that B occurs immediately after A.
2. **Directly Precedes (←)**: Activity A is directly preceded by Activity B in the event log.
    - Example: A ← B means that A occurs immediately after B.
3. **Concurrent (||)**: Activities A and B can occur in parallel, meaning there is no fixed order between them.
    - Example: A || B means that A and B can happen simultaneously or in any order.
4. **Never Follows (#)**: Activity A never follows Activity B and vice versa.
    - Example: A # B means that A and B never occur immediately after each other.
5. **Not Co-occurring (×)**: Activities A and B do not co-occur in the same case.
    - Example: A × B means that if A happens, B does not happen in the same case, and vice versa.

Example of a Footprint Matrix

Let's consider a process with activities A, B, C, and D. The footprint matrix for this process could look like this:

	A	B	C	D
A	#	→		
B	←	#	→	#
C				←
D	→	#	←	#

Interpretation

- **A → B**: Activity B directly follows Activity A.
- **A || C**: Activities A and C can occur concurrently.
- **A ← D**: Activity A is directly preceded by Activity D.
- **B # D**: Activities B and D never follow each other.
- **B → C**: Activity C directly follows Activity B.

Usage

- **Process Discovery**: Helps in identifying the structure of the process by revealing the order of activities.
- **Conformance Checking**: Compares the expected model with the actual event log to identify deviations.
- **Performance Analysis**: Analyzes the performance of different activities and transitions within the process.

The Footprint Matrix provides a clear and concise way to understand the flow and relationships between activities in a business process.

▼ Exercice

Question 1

1/1 point (ungraded)

Consider the following event log. Which of the footprint matrix is correct based on this event log?

15: a b c e d

13: b a d e d

1: a b d e c

	a	b	c	d	e
a	#	||	#	→	#
b	||	#	→	→	#
c	#	←	#	#	→
d	←	←	#	#	||
e	#	#	←	||	#

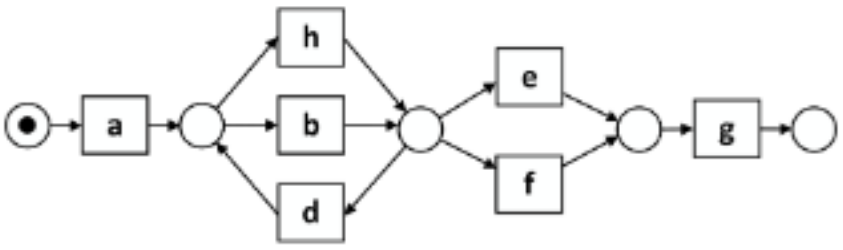
	a	b	c	d	e
a	#	→	→	→	#
b	←	#	→	→	#
c	←	←	#	#	→
d	←	←	#	#	→
e	#	#	←	←	#

	a	b	c	d	e
a	#	||	#	→	#
b	||	#	→	→	#
c	#	←	#	#	||
d	←	←	#	#	||
e	#	#	||	||	#

Question 2

1/1 point (ungraded)

Consider the following process model. Which footprint matrix is correct based on this model?



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

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

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

Question 3

1/1 point (ungraded)

Consider the following footprint matrices that are extracted from an event log and a process model. Calculate the footprint-based conformance.

Event log footprint matrix:

	a	b	c	d
a	#	#	→	→
b	#	#	||	→
c	←	||	#	||
d	←	←	||	#

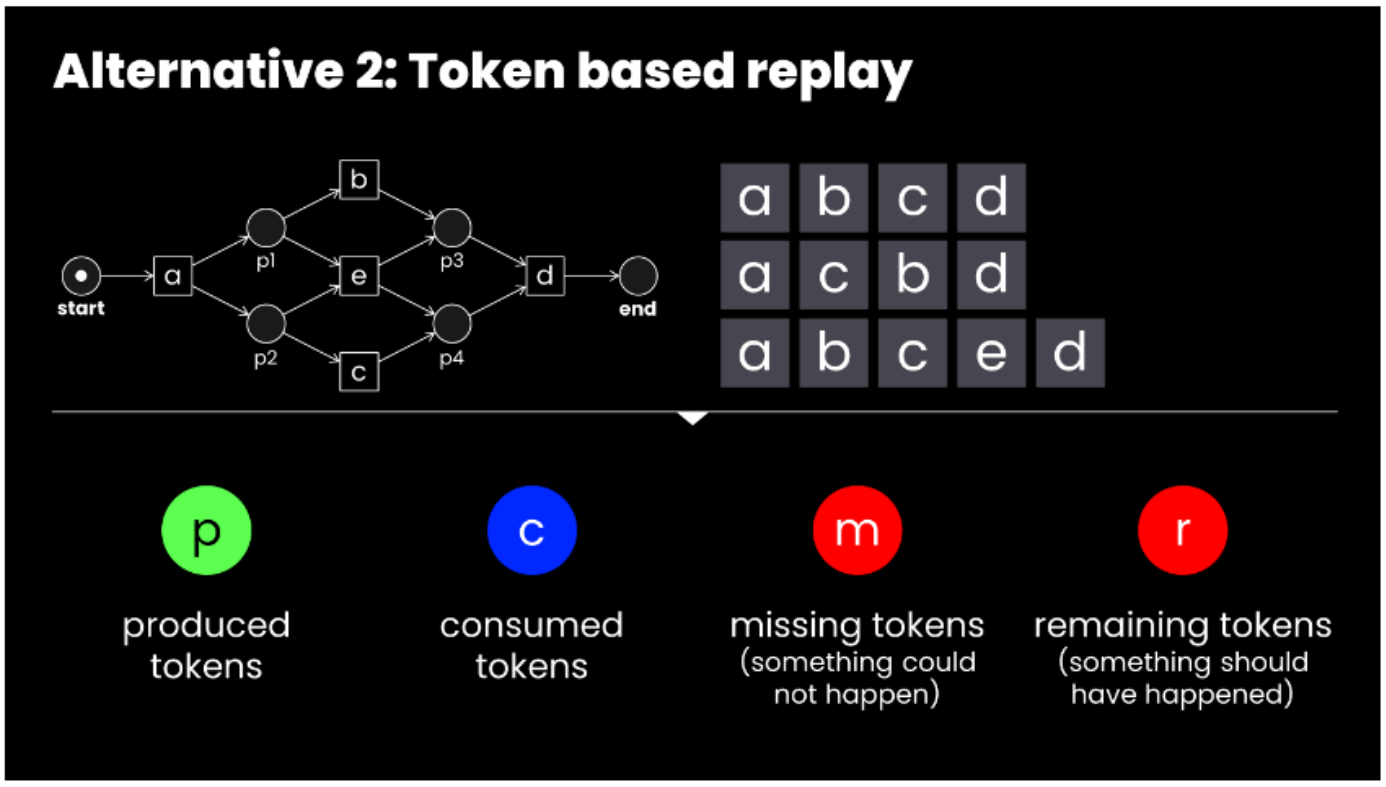
Process model footprint matrix:

	a	b	c	d
a	#	#	→	→
b	#	||	→	→
c	←	←	#	||
d	←	←	||	#

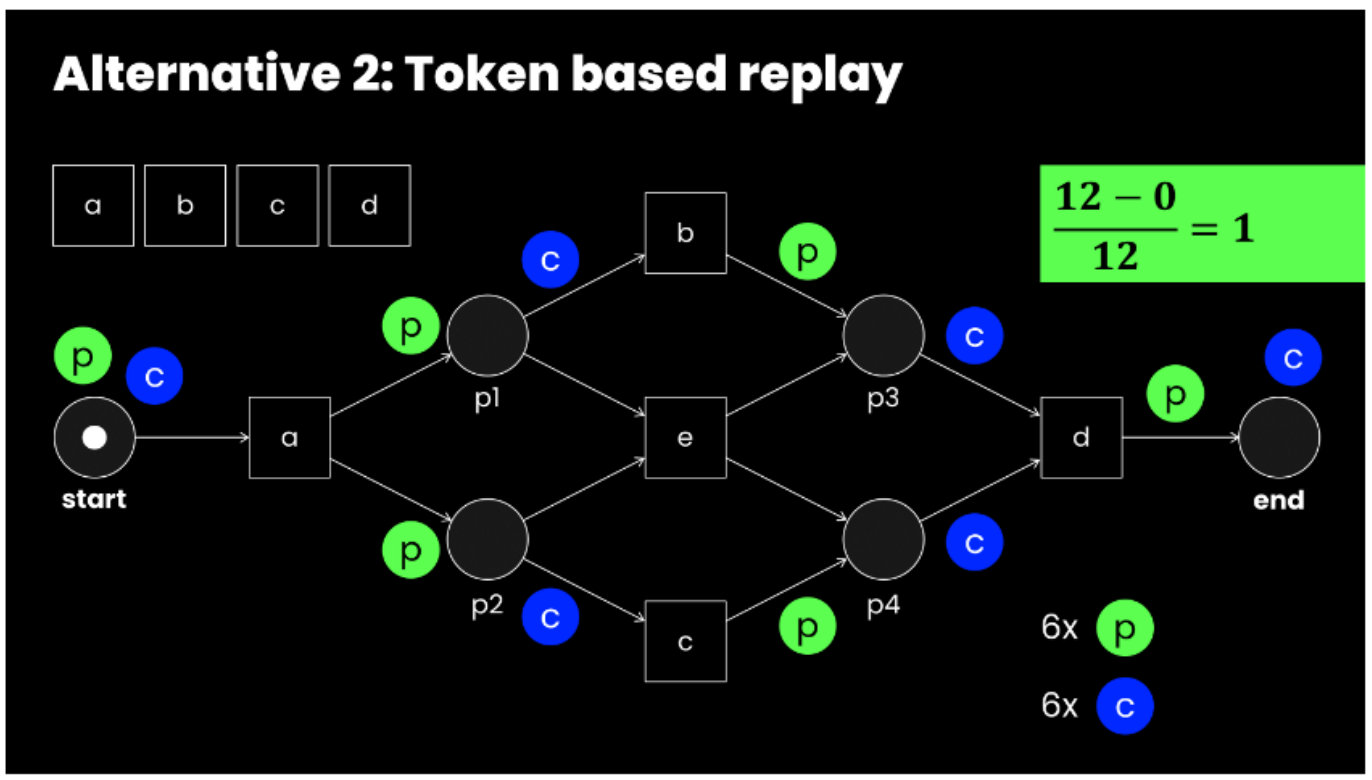
- ○ 0.75
- ○ 0.9375
- ○ 0.875
- ◉ 0.8125 ✔

▼ **Token-Based Replay Conformance Checking**

Next to computing alignments and comparing footprints, it is possible to conduct *token-based replay*. This is more scalable than alignments, but less precise, because it does not produce paths through the model. However, the diagnostics reveal compliance-related pain points. The idea of token-based replay is to simply try and replay the trace on the Petri net model. Events correspond to the occurrence of Petri-net transitions. If a transition is not enabled, then we add the tokens needed. These are called *missing tokens*. If at the end, tokens remain in the Petri net while reaching the end of the trace, these are *remaining tokens*. Initially, a token is produced for the initial place to initialize the process model. At the end of the trace, a token is removed from the final place. If it is not there, a missing token is added.



Consider the trace $\langle a, b, c, d \rangle$. If we replay this trace, there are no missing or remaining tokens. In total, there are six produced and six consumed tokens, as indicated below. Hence, fitness is equal to $\frac{(6+6)-(0+0)}{6+6} = 1$. In general, fitness can be defined as $\frac{(p+c)-(r+m)}{p+c}$. A fitness value of one implies that the trace was replayable without any problems.



Next, we consider the trace $\langle a, b, c, e, d \rangle$. If we replay this trace, there are two missing and two remaining tokens. In total, there are eight produced and eight consumed tokens. Hence, fitness is equal to $\frac{(8+8)-(2+2)}{8+8} = 0.75$.

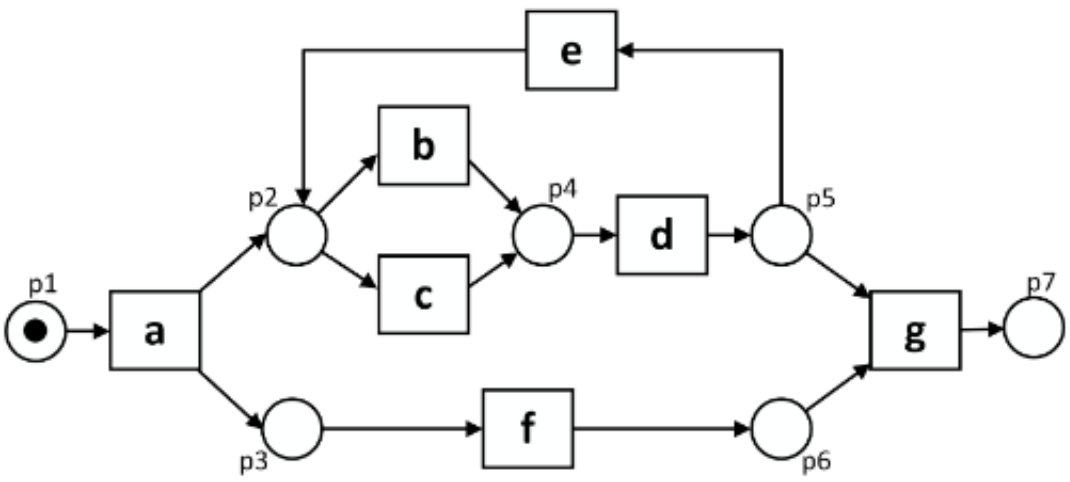
▼ **Exercice**

Question 1

1/1 point (ungraded)

Consider the following event log and process model. Apply the token-based replay and calculate the number of produced, consumed, missing, and remaining tokens in each place. Which of the following statements are correct?

Process model:



Event log:

1: a c d e b f g

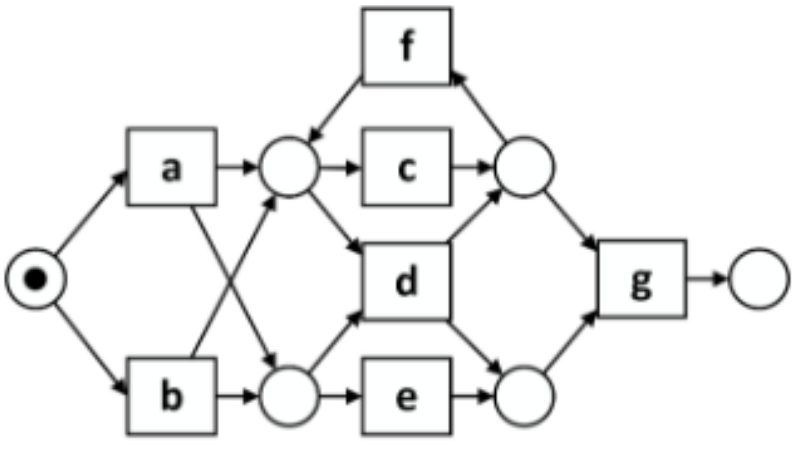
- [ ] Two tokens are remaining in place p4.
- [ ] Two tokens are produced in p5 but only one of them is consumed.
- [x] Two tokens are produced and consumed in p2.
- [ ] Two tokens are produced in p6 but only one of them is consumed.

Question 2

1/1 point (ungraded)

For the following process model and event log, calculate the number of produced, consumed, missing, and remaining tokens.

Process model:



Event log:

1: a b c g

The total number of produced tokens is:

7 ✔

7

The total number of consumed tokens is:

6 ✔

6

The total number of missing tokens is:

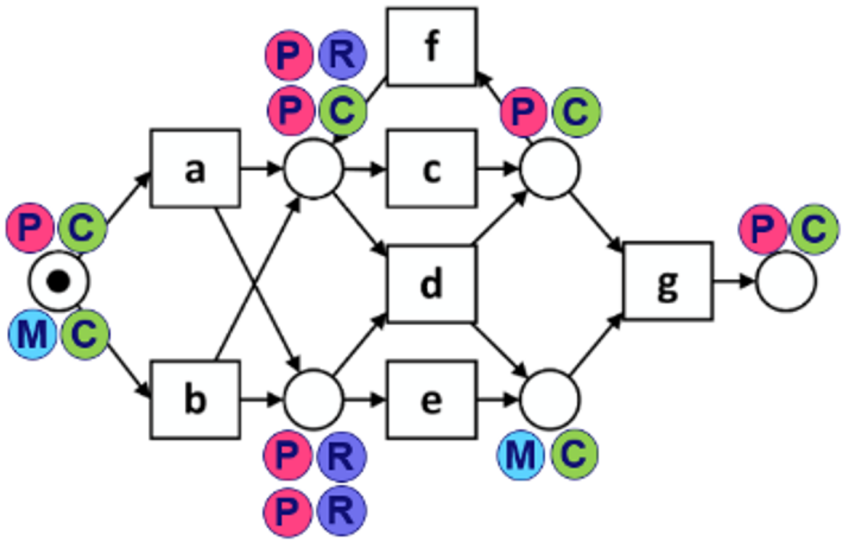
2 ✔

2

The total number of remaining tokens is:

3 ✔

3



Question 3

1/1 point (ungraded)

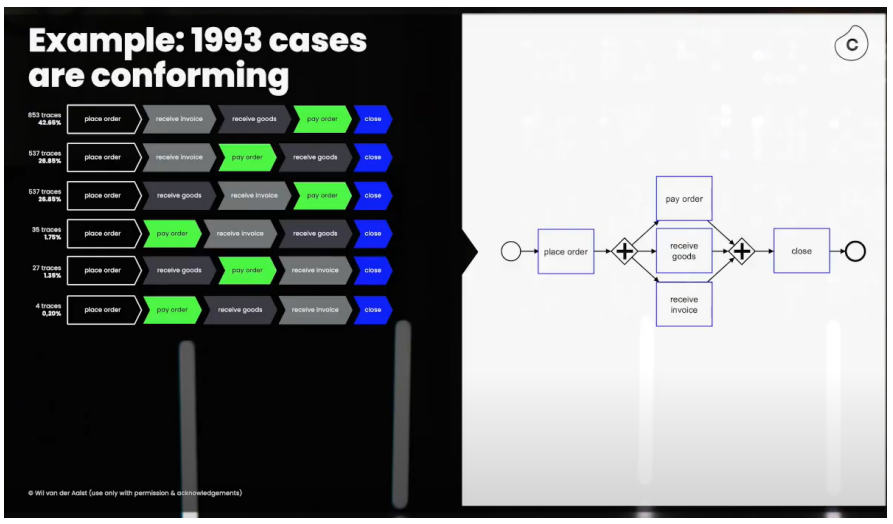
For the following numbers calculate the token-based replay fitness.

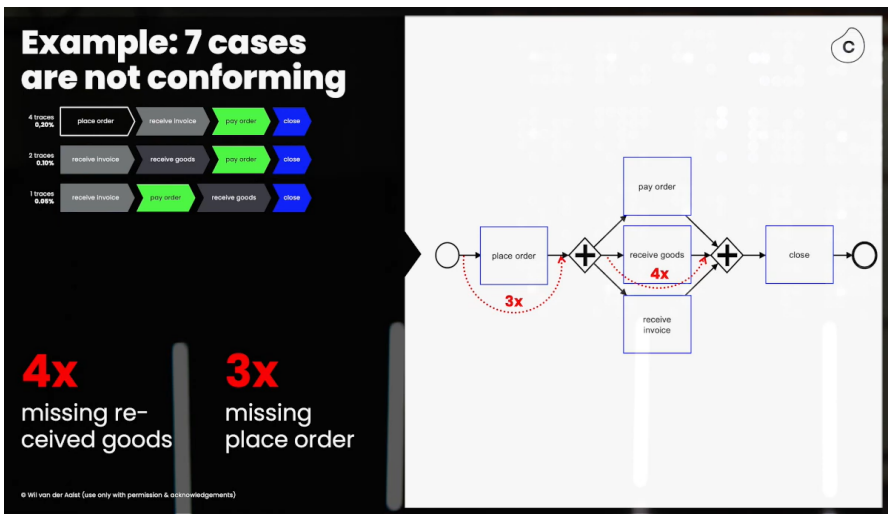
Produced tokens = 8, Consumed tokens = 6, Missing tokens = 2, Remaining tokens = 3

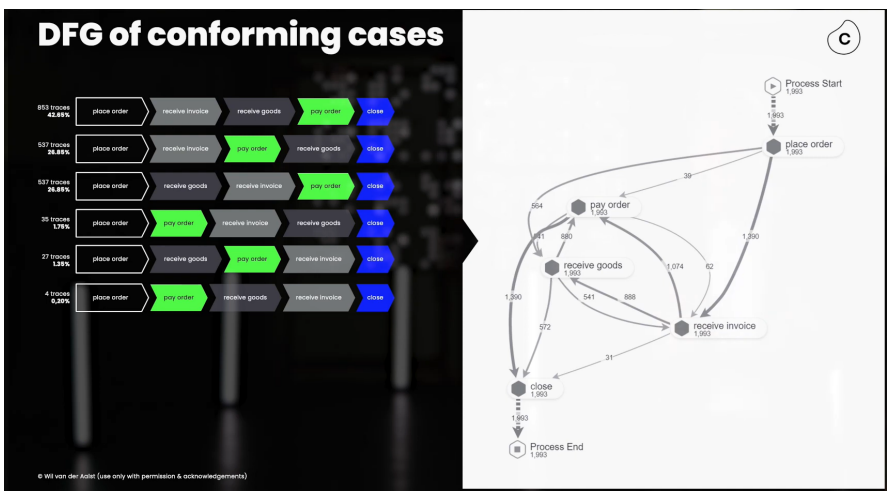
- 0.42
- **0.64** ✔
- 0.72

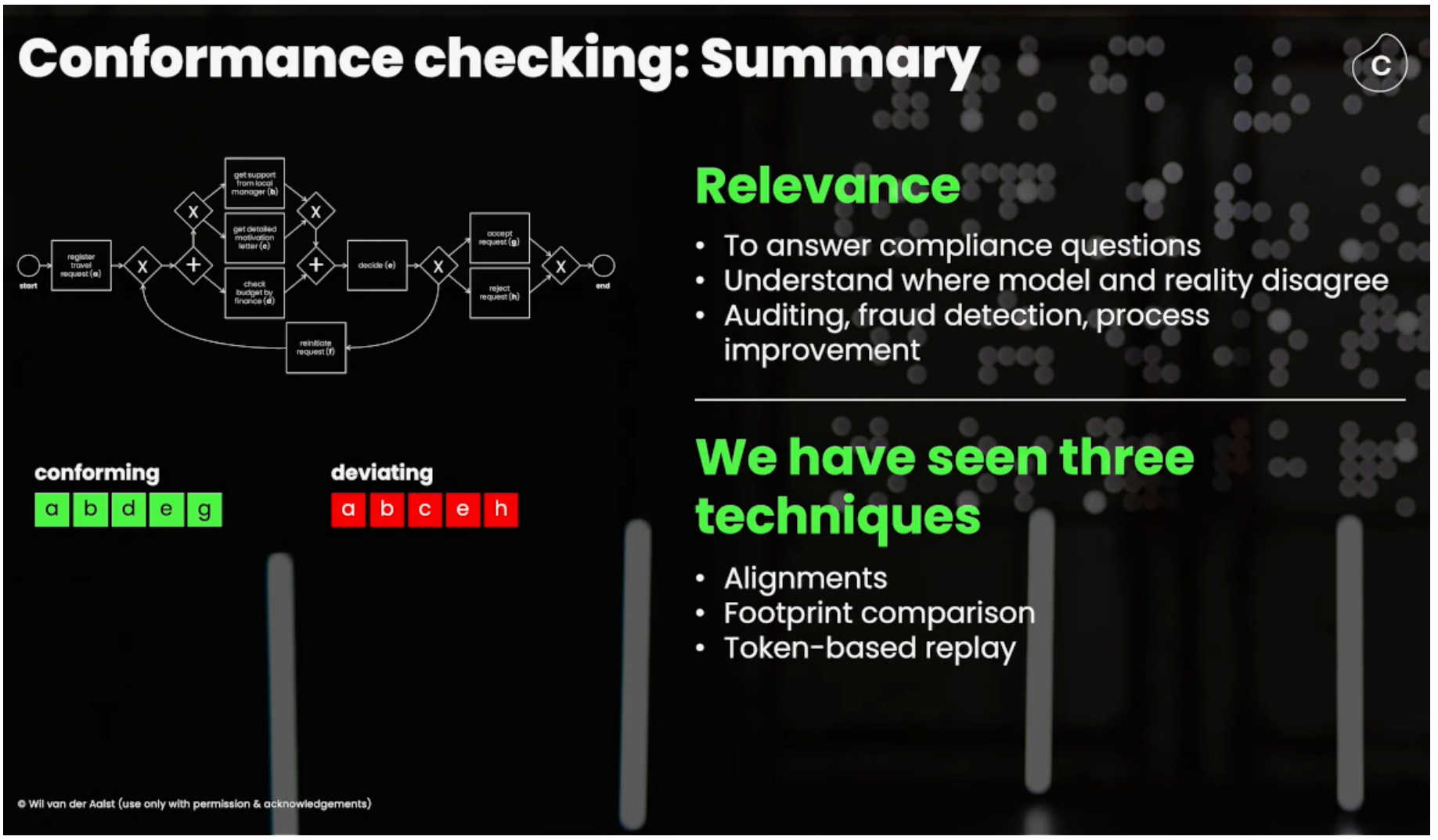
▼ Summary

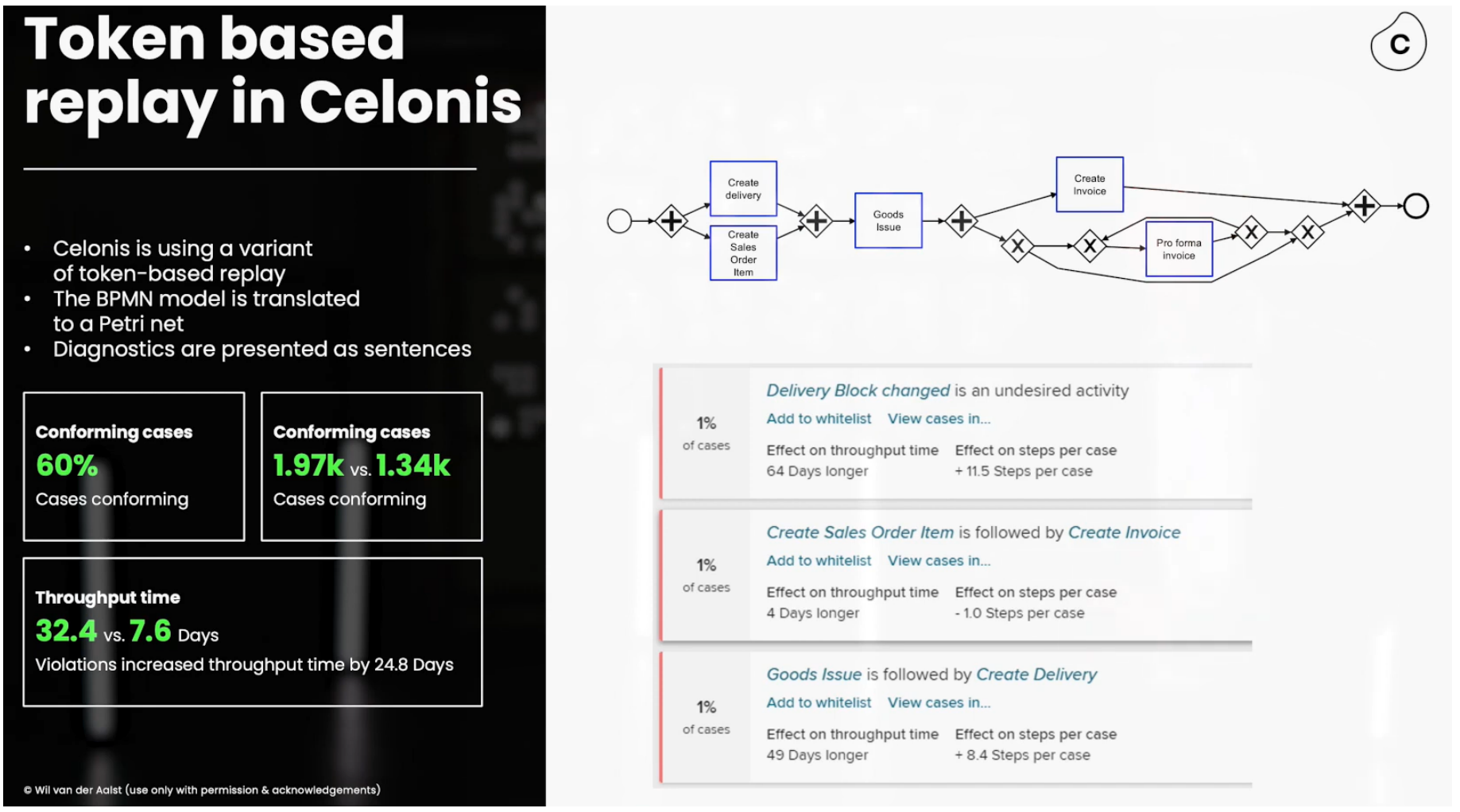
https://www.youtube.com/watch?v=FNvDls3Ru0Y











▼ Celonis - Conformance Checking

the different ways of how to obtain a normative process model. You can either design your process model from scratch in the Celonis BPMN modeler, upload an existing BPMN model or even mine a BPMN model from your data. Celonis then flags process violations where your as-is process deviates from the normative process model. You can find the data set we worked with again below for your own reference.

You can find the data for this tutorial, together with a tutorial on how to upload data, in the new Celonis interface below:

- https://courses.edx.org/assets/courseware/v1/821d3e9cec3fc8a0eb43e57503a8cc09/asset-v1:RWTHx+PM+1T2024d+type@asset+block/concurrency_matters_dev.csv
- https://www.youtube.com/watch?v=rb4BJ0trbx8