UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Maksym Melnyk

Data-aware Conformance Checking

Master’s Thesis (30 ECTS)

Supervisors: Marlon Dumas
Thomas Baier

Tartu 2019

Data-aware Conformance Checking

Abstract:

Conformance checking is one of the most common tasks in the field process mining. The goal of conformance checking is to compare a process model against an event log in order to quantify or describe how the behavior recorded in the log deviates with respect to the behavior captured by the process model. Most of the existing conformance checking techniques focus on the control-flow perspective. In this thesis, we propose a conformance checking technique that takes into account the data perspectives in addition to the control-flow perspective. The proposed approach is implemented as a tool that takes as input a BPMN process model and an event log. The tool has been implemented using the Elixir programming language. The thesis also reports on a performance evaluation of the proposed approach.

Keywords:

Process Mining, Conformance Checking, Data-aware Conformance Checking, BPMN, Event Log, Automaton, Elixir.

CERCS:

P170 – Computer science, numerical analysis, systems, control.

Andmeteadlik Vastavuse Kontrollimine

Lühikokkuvõte:

Vastavuse kontrollimine on üks kõige tavalisemaid ülesandeid protsessikaeve valdkonnas. Vastavuse kontrollimise peamine eesmärk on kontrollida protsessimudeli vastavust sündmuste logidele selleks, et hinnata või kirjeldada kuidas registreeritud käitumine protsessimudelis kirjeldatud käitumisest erineb. Enamus olemasolevatest vastavuse kontrollimise tehnikaid põhineb kontrollvoolu perspektiivile. Käesolev lõputöö pakub välja tehnika, mis lisaks kontrollvoolule põhinevale tehnikale arvestab ka andmete perspektiivile. Väljapakutud lähenemisviis on implementeeritud tarkvaralise lahendusena, mis kasutab sisendiks BPMN mudelit ja sündmuste logi. Loodud tarkvara töörist on loodud kasutades programmeerimiskeelt Elixir. Lõputöö sisaldab samuti ka välja töötatud lahenduse tulemuslikkuse hinnangut.

Võtmesõnad:

Protsessikaeve, Vastavuse kontrollimine, Andmeteadlik vastavuse kontrollimine, BPMN, Sündmuste logid, Automatiseerimine, Elixir.

CERCS:

P170 – Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine.

Table of Contents

1 Introduction

Nowadays, the competition in all industries is very high. Some companies are value-oriented, some of them are price oriented. Companies try to become more efficient in many different ways and one of them is by improving their business processes. Process Mining is a set of techniques that helps companies achieve this goal.

Process Mining is a very popular topic, and it makes sense, processes are becoming very complex and the amount of data is growing very fast. It is not possible anymore to improve processes in “MS Excel” or other similar programs because they are not suitable to deal with a large amount of data and with very complex processes. This is why many companies started to look for other possible solutions since even small improvement can potentially save millions of dollars.

Process Mining has two main tasks: Process Discovery and Conformance Checking. Although these tasks are very connected between each other, in this paper we will concentrate on Conformance Checking task since it has many different sides and edges.

In order to improve any business process, we need to know where is the problem or place that can be improved. By knowing that we can make some changes in the process that can save our money. If the business management is smart enough, they will design a process before running it. But nothing is perfect in the real world and the real-life process is usually not the same as the designed one for many different reasons. One of the reasons could be the impossibility to make one action before another or necessity to skip an action. By using Conformance Checking we can find these deviations between the real process and designed one and by using this information make changes to the designed process.

In most of the literature regarding Conformance Checking, authors focused on the control-flow perspective, which means, they care only about the correct ordering of the actions. In a real-life process, however, other perspectives are important and some other aspects can affect the process. Multi-perspective Conformance Checking is aware not only of control-flow but also about data, time and resources. Since traditional Conformance Checking techniques do not take into considerations those perspectives, many deviations remain undetected. And this an important issue because if we do not have full information about our process and its deviations, we cannot make good assumptions for improving it.

In this thesis we will provide an example of processes where some deviations remain undetected until we add data perspective to the process model. To address this gap, we present a conformance checking technique that takes into account the data perspective.

In order to design processes and unify its format, the community created the Business Process Model and Notation (BPMN). There are many implementations of this format in many different languages. However, there is no implementation for Elixir, which is becoming very popular nowadays because of its performance, simplicity and functional approach.

From a tooling perspective, the main contribution of this thesis is a parser for BPMN and a conformance checking tool written entirely in Elixir. Both the parser and the algorithm take into account the control-flow and the data perspectives.

The thesis also reports on some performance tests to assess the efficiency of the proposed conformance checking technique on different logs and models.

2 Background

In this section, we describe needed terms and definitions in order to fully understand the thesis topic. At the end, we provide a running example for better understanding.

2.1 Event Logs and Traces

In this subsection, we will describe definitions that are needed to understand how companies observe and store their process executions.

Each company that do enterprise solutions has a lot of processes. All of the events of the existent processes have to be saved in some kind databases or files. Some companies decided to design a database for this purpose, some of them just keep all of the events happened in the files. Unfortunately, most of the companies have their own format to keep all of this data. However, some attributes remain the same.

Definition 1 (Event). *An event $e \in B(A)$ is execution of an activity from the set of activities A with mandatory attributes:*

1. *Trace ID (will be described later)*
2. *Process activity ID*
3. *Start timestamp*
4. *Finish timestamp*

There could be a different number of custom attributes that the company decided to keep. It is usually “Executor”, “Role” etc.

In simple words, an event is an atomic structure in the process that describes when one particular activity from the process happened and how long it has been executed.

Definition 2 (Event Log). *An event log $L \in B(A)$ is a set of events e over the activity set A.*

If we consider event log as a file, then it will be *CSV* file where columns are event attributes and rows are events.

Usually, there are separated event logs for each process (there could be many event logs for one process) but if the company decided to have one log for many processes then each event should have additional attribute "Process ID". However, this type of logs is very rare, and we will not consider it in this thesis.

Trace	Activity	Start	End	Role
1	Incident logging	2016/01/04 12:09:44	2016/01/04 12:09:44	Agent
1	Incident classification	2016/01/04 12:10:44	2016/01/04 12:17:44	Agent
1	Initial diagnosis	2016/01/04 12:34:44	2016/01/04 12:39:44	Agent
1	Functional escalation	2016/01/04 12:41:44	2016/01/04 12:48:44	Agent
1	Investigation and diagnosis	2016/01/04 18:13:44	2016/01/05 01:36:44	Special Agent
1	Resolution and recovery	2016/01/05 03:56:44	2016/01/05 04:30:44	Agent
1	Incident closure	2016/01/05 04:31:44	2016/01/05 04:48:44	Agent
2	Incident logging	2016/01/04 13:09:44	2016/01/04 12:09:44	Agent
2	Initial diagnosis	2016/01/04 13:10:44	2016/01/04 12:17:44	Agent
2	Incident classification	2016/01/04 13:34:44	2016/01/04 12:39:44	Agent
2	Functional escalation	2016/01/04 13:41:44	2016/01/04 12:48:44	Agent
2	Investigation and diagnosis	2016/01/04 19:13:44	2016/01/05 01:36:44	Special Agent
2	Resolution and recovery	2016/01/05 04:56:44	2016/01/05 04:30:44	Agent
2	Incident closure	2016/01/05 05:31:44	2016/01/05 04:48:44	Agent

Figure 1: Event log example (Incident Management).

Definition 3 (Trace). *A trace* $t = < e_1,\ e_2, \dots e_n >$ *is a sequence of events that are related to the one particular process execution. The sequence is ordered by event timestamps.*

As mentioned before, a trace is a sequence of events. Practically speaking, we can retrieve all the traces from the event log by grouping it by "Trace ID" (this attribute can be called "Case ID" as well). If we look at the example of the event log from Figure 1, we can see that all the events that are related to the trace with number 1 and it is ordered by timestamps. So, roughly speaking, we can consider that as a trace.

Traces, as well as events, can have attributes specific to one particular process execution. For example, the trace can have attribute "County" with value "Germany" which means, according to the incident management example (Figure 1), that the incident has happened in Germany. Of course, it doesn't make sense to keep this attribute for an event because it is the same for all events in the trace. Companies decide which attributes they need to have in order to do better postprocessing of the logs, that is why they are different from process to process and from log to log. Trace attributes are usually stored in separated from event log files or database tables.

2.2 BPMN models

In this subsection, we provide definitions with appropriate examples which are useful in business process designing and management overall.

As it was mentioned in the introduction section of this thesis, if a company is smart enough and has good management, before running new business process, it would be good if the company design this process using some special tools.

Decades before, companies draw and design processes on a paper or, if a company has enough resources and money, using some proprietary programs and tools. But in the end, business understood that it is good to have some unified business process format in order to share it not only inside the organization but also for business cooperation etc.

In 2011 Object Management Group designed a new modelling standard called Business Process Model and Notation (BPMN). BPMN quickly became very popular because of its complete range of supported abstractions that is understandable for all type of business users.

Definition 4 (BPMN model). *A process model that conform BPMN requirements.*

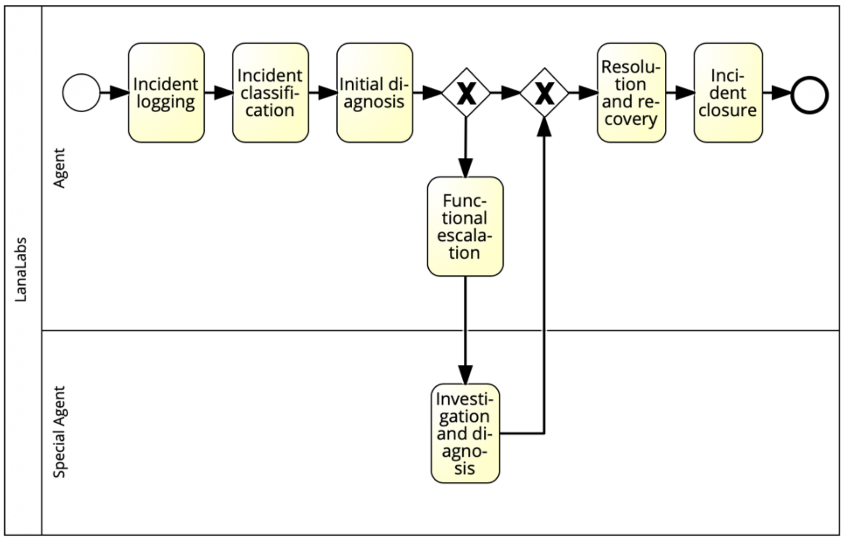


Figure 2: BPMN model example, visual representation (Incident Management).

We need to remember that BPMN is just a format, that is why there are a lot of tools that implement this format. These tools, for example, “Camunda”, “Signavio”, usually provide a possibility to create BPMN models via graphical interface as shown in Figure 2. These BPMN models then can be saved to one of the text, usually *XML*, formats (Figure 3).

```
<bpmn:process id="incident_management" isExecutable="false">
  <bpmn:startEvent id="start_event">
    <bpmn:outgoing>sequence_flow_1</bpmn:outgoing>
  </bpmn:startEvent>
  <bpmn:task id="incident_logging" name="Incident logging">
    <bpmn:incoming>sequence_flow_1</bpmn:incoming>
    <bpmn:outgoing>sequence_flow_2</bpmn:outgoing>
  </bpmn:task>
```

Figure 3: BPMN model example, text representation in xml format (Incident Management).

Even though BPMN provides a wide range of abstractions, working with data is not as good as we want. Of course, there are data objects, but they are not more than just graphical presentation. If we want to use the data, we need a different level of abstraction. Fortunately, the Camunda (https://camunda.com) modelling tool provides an extension mechanism, using this mechanism we can extend BPMN format with the needed abstractions. Camunda has an extension for adding conditions to sequence flows and input/output parameters to activities. With these abstractions, we can easily include data to our BPMN models and then use it for Conformance Checking.

2.3 Conformance Checking

Conformance Checking is a family of techniques to compare process model with an event log of the same process model.

In simple words, having a BPMN process and an event log we can use Conformance Checking to answer the question: how close a real-life process with a modelled one?

If we look at the process model example from Figure 2 and analyze it, we can say that Trace 1 from Figure 1 perfectly fit the model. So that there are no deviations for this trace.

If we try to align Trace 2 from Figure 1 to the process model from Figure 2, we can see that the trace doesn't fit the model. After further analysis, we find out that activities "Incident classification" and "Initial diagnosis" are switched. So, there is a deviation for this trace.

This example of control-flow conformance checking shows us that this type of conformance checking works if there are misplaced activities in respect to the process model.

Control-flow conformance checking is based only on ordering of events/activities. It is the most popular type of conformance checking and most of the papers about conformance checking based on this type.

Control-flow conformance checking does not take into account the timestamps (time perspective), it only takes into account the order of the events in each case but not the actual time of occurrence. Also, control-flow conformance checking as illustrated above does not take into account any data conditions that may be attached to the XOR-split gateways of the process model (data perspective), it doesn’t take into account the (human) resources who perform the activities in the process (resource perspective).

2.4 Conformance Checking Tools

Conformance Checking is implemented is several opensource and commercial tools including Apromore (apromore.org), Signavio Process Intelligence (signavio.com), Celonis (celonis.com) and LanaLabs (lanalabs.com).

This thesis has been written in partnership with LanaLabs. LanaLabs is a process mining startup based in Berlin. They provide several process mining tools. In this subsection, we describe *Lana Process Mining Tool* and we give a short overview of its features.

2.4.1 Process Discovery

The first thing we need to do in order to use the tool is to upload an event log. We use “Incident Management” event log as an example that we have been using in this thesis. After uploading an event log, we already can view a discovered process map of this log (Figure 4). Discovered process map is a graph where nodes are activities and arcs represent the direct following relation between two activities.

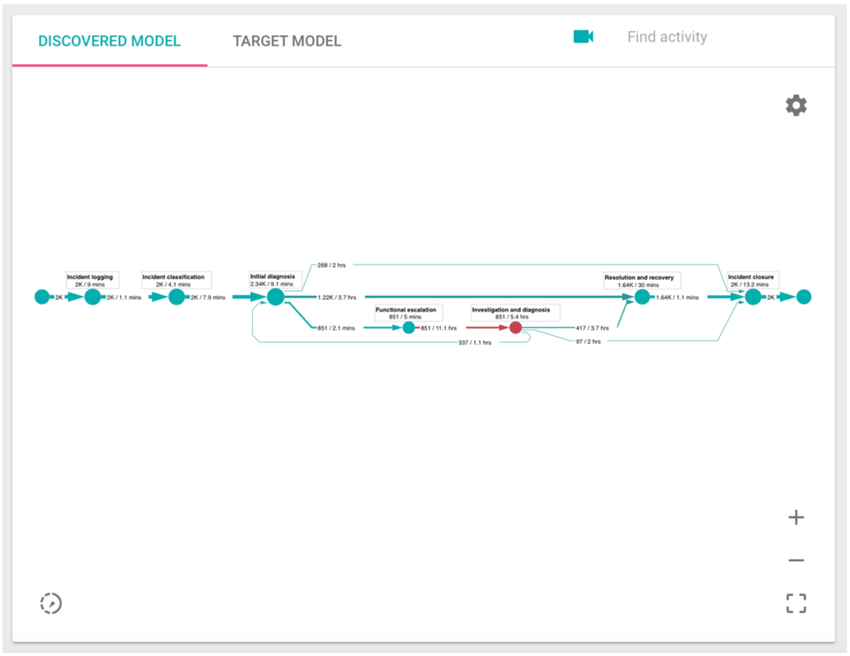

DISCOVERED MODEL
TARGET MODEL
Find activity
Incident logging 2K / 9 mins
Incident classification 2K / 4.1 mins
Initial diagnosis 2.34K / 9.1 mins
Functional escalation 851 / 5 mins
Investigation and diagnosis 851 / 5.4 hrs
Resolution and recovery 1.64K / 30 mins
Incident closure 2K / 13.2 mins
2K
2K / 1.1 mins
2K / 7.9 mins
268 / 2 hrs
1.22K / 3.7 hrs
851 / 2.1 mins
851 / 11.1 hrs
417 / 3.7 hrs
97 / 2 hrs
337 / 1.1 hrs
1.64K / 1.1 mins
2K

Figure 4: Incident Management discovered process map.

We can see some base statistics like frequency of occupancies of the events and some weak points.

2.4.2 Conformance Checking

In order to do conformance checking, we need to upload a BPMN model or create a new one from scratch in the tool (Figure 5).

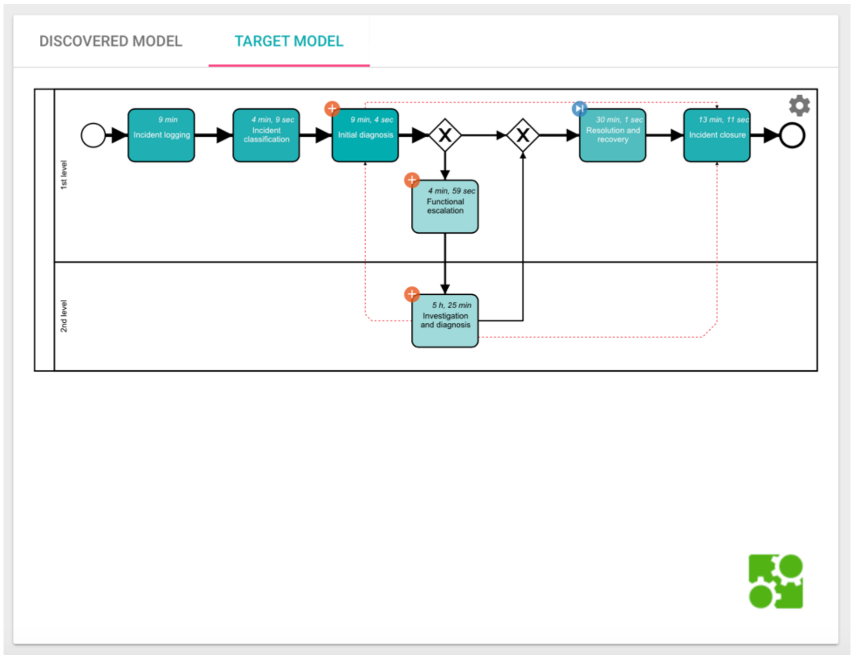
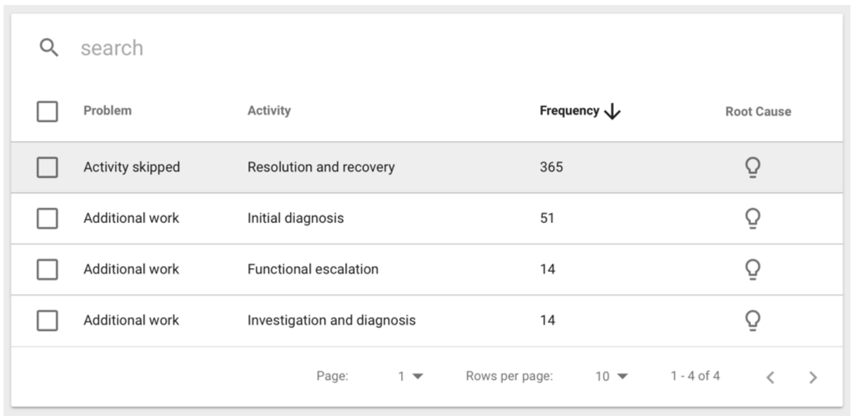


Figure 5: Target model of the Incident management example.

As we can see the tool already shows us deviation paths and skipped and inserted activities (*move-on-log* and *move-on-model*).

We can look into the deviations in detail on "Action" page (Figure 6).


search

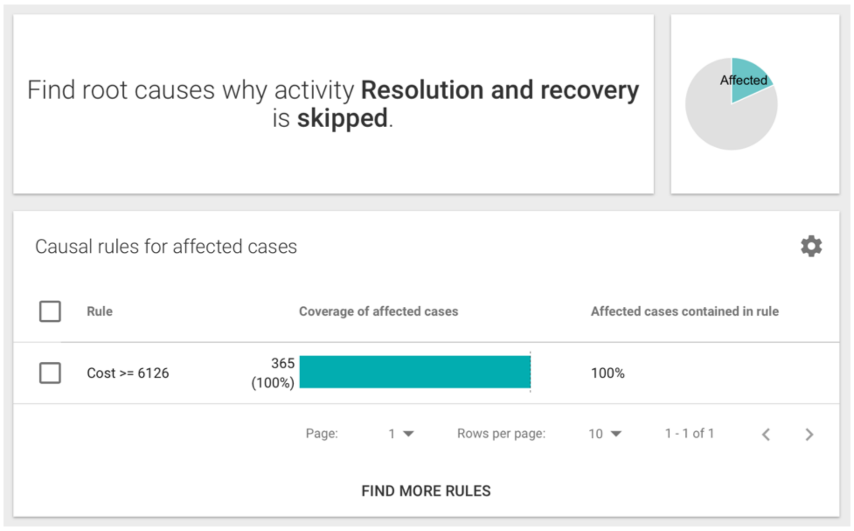
	Problem	Activity	Frequency	Root Cause
☐	Activity skipped	Resolution and recovery	365	
☐	Additional work	Initial diagnosis	51	
☐	Additional work	Functional escalation	14	
☐	Additional work	Investigation and diagnosis	14	

Page: 1 Rows per page: 10 1 - 4 of 4

Figure 6: Incident Management deviations.

2.4.3 Root cause analysis

As we can see in Figure 6, we can do root cause analysis on each of the deviations by clicking on the special button. Let's, for example, run root cause analysis on skipping "Resolution and Recovery" activity.


Find root causes why activity **Resolution and recovery** is **skipped**.

Causal rules for affected cases

	Rule	Coverage of affected cases	Affected cases contained in rule
☐	Cost >= 6126	365 (100%)	100%

Page: 1 Rows per page: 10 1 - 1 of 1

FIND MORE RULES

Figure 7: Root cause analysis example.

There is a possibility to run root cause analysis in any part of the application and on different combinations deviations.

2.4.4 Statistics

As described above we can view conformance checking statistics in various places. There is, however, the "Statistics" page where we can see a full picture of the conformance checking results and even more (Figure 8).

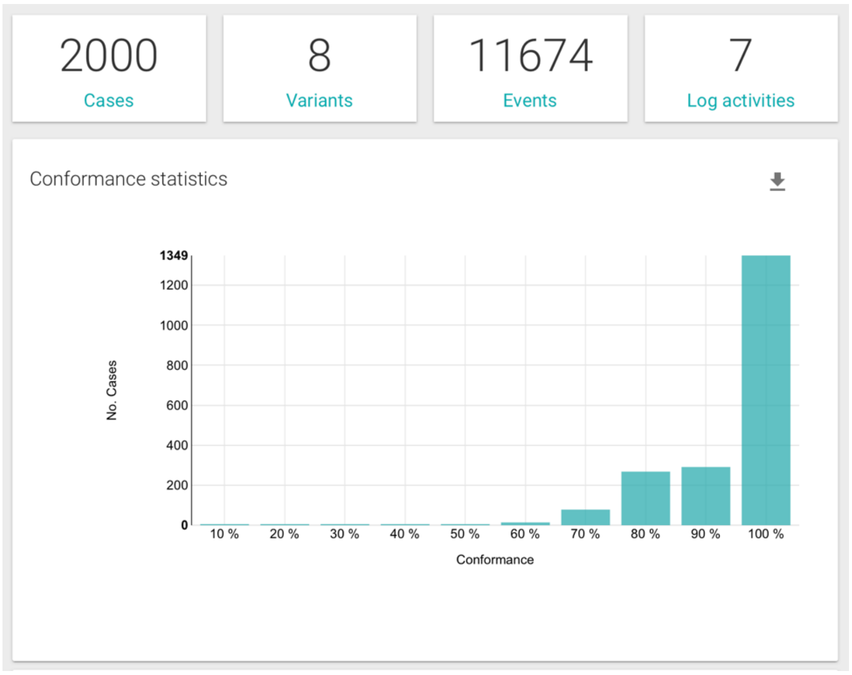


Figure 8: Conformance statistics page.

In this subsection, we described only a few features of the *Lana Process Mining Tool*. There are way more features like filtering event logs, creating dashboards etc. You can read more and try the tool on the website https://lanalabs.com.

We saw some nice visualizations, histograms etc. In order to create those visualizations, we need to compute the alignment between each trace and the process model. The concept of alignments and how to compute them is described in Section 3 of this thesis.

2.5 Multi-Perspective Conformance Checking

In this subsection, we describe multi-perspective conformance checking in respect to time, resource and data perspectives.

2.5.1 Time perspective

When considering real-life models, it may happen that an activity should be executed after a certain period of time (timer events). If we do not take the time perspective into account, we cannot check when activity was executed. However, in real life this is important and if an activity "did not meet the deadline" we should consider it as a deviation. Consider, for example, the model in Figure 9 taken from [2]:

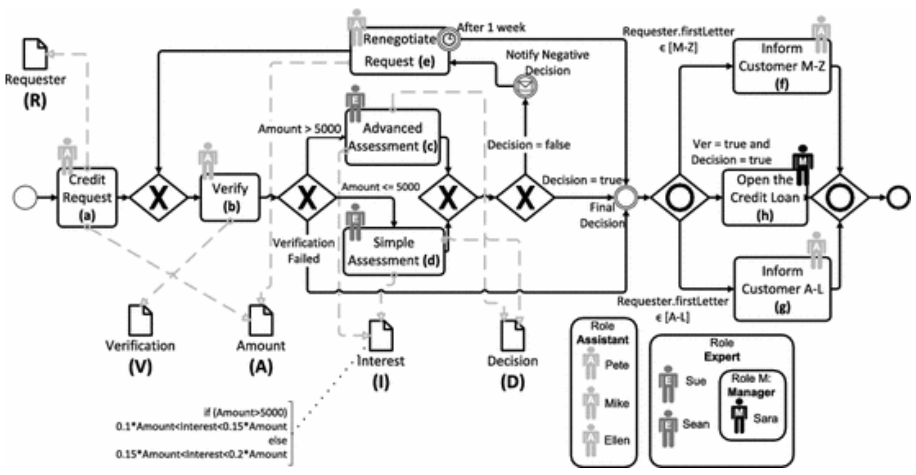


Figure 9: Process model describing a process to handle credit requests.

As we can see from the Figure 9, "Renegotiation Request" should be processed within one week. This constraint may be a cause of a deviation that cannot be found using simple control-flow conformance checking.

2.5.2 Resource Perspective

In process models, each activity is usually associated with a particular role. It could be one exact resource as well as a group of resources. This means that an activity associated with a resource must be executed by that resource.

If we look at the process model from Figure 2, the activity "Investigation and Diagnosis" is associated with the resource "Special Agent". If the activity is performed by another resource, for example, "Agent", this is a deviation, which is not catchable by control-flow conformance checking.

As a conclusion of this subsection, we can say that the resource perspective for conformance checking is important and must be considered so that more deviations can be identified.

2.5.3 Data Perspective

In this subsection, we will introduce the data perspective, one of the most important aspects of conformance checking.

As it was already mentioned, we can add conditions to XOR splits. This is the simplest example of data perspective. However, more complex scenarios can come up. In process models, the results of activities can be stored in variables and this information can be used as input for other activities. For example, in Figure 9, we can see that the choice between “Simple assessment”, “Advanced assessment” and skipping assessment is made considering the “Amount” and “Verification” variables, which are the results of “Credit request” and “Verify” activities. If a decision is made against the rules, we must identify a deviation that is not possible to identify when using only the control-flow perspective.

Moreover, considering the real world where the business logic is very complex, it is possible to build decision models based on business rules. A decision model could be represented as a Decision Model and Notation (DMN) diagram. DMN is a standard designed to work with BPMN process models, so it is possible then to connect a DMN model to the BPMN model and, after the analysis, we can get more accurate results.

In this thesis, we will not consider very complex business logic and rules. We will focus only on simple cases like adding conditions to XOR gateways. In Section 3 we will go behind the Conformance Checking with data perspective by providing a running example of this case.

Let’s then slightly change the model as shown in Figure 2:

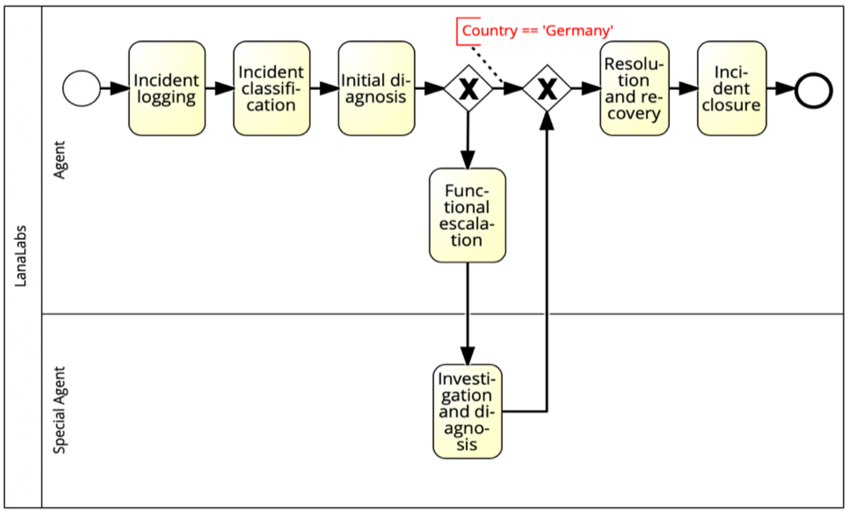


Figure 10: Incident Management process model (with data perspective)

We can see that the process model from Figure 10 now has additional elements. One of the sequence flows has a condition. By adding conditions, we apply data perspective to the process model. Now, in order to do “Functional escalation” and “Investigation and diagnosis” the country of the incident must be different from Germany. If we do those activities for the incident in Germany it should be considered as a deviation.

In order to have a full picture let’s also consider the trace attributes for the traces as shown in Figure 11:

Trace	Country
1	Germany
2	Germany

Figure 11: Incident Management trace attributes.

If we try to align the Trace 1 from Figure 1 to the new model from Figure 10 (taking into account the trace attributes from Figure 11), we will see that the trace doesn’t fit the model anymore because of the sequence flow condition. If we don’t take into account the data, the trace still fit the model perfectly, however, we need to consider data in order to catch all the deviations.

Let’s consider a new trace shown in Figure 12:

Trace	Activity	Start	End	Role
3	Incident logging	2016/01/04 12:09:44	2016/01/04 12:09:44	Agent
3	Incident classification	2016/01/04 12:10:44	2016/01/04 12:17:44	Agent
3	Initial diagnosis	2016/01/04 12:34:44	2016/01/04 12:39:44	Agent
3	Resolution and recovery	2016/01/05 03:56:44	2016/01/05 04:30:44	Agent
3	Incident closure	2016/01/05 04:31:44	2016/01/05 04:48:44	Agent

Figure 12: Incident Management trace example without any type of deviations.

With the following trace attributes shown in Figure 13:

Trace	Country
3	Germany

Figure 13: Incident management trace attributes for the trace from Figure 12.

Considering trace from Figure 12 and trace attributes from Figure 13, we can see that this trace perfectly fit the model (Figure 10) from both control-flow conformance checking and data-aware conformance checking.

As a conclusion of this subsection, we saw that there is some type of deviations that control-flow conformance checking cannot catch, however data-aware conformance checking can deal with all the data related deviations.

3 Related Work

In this section, we describe related works about Conformance Checking. We will focus on Conformance Checking with multiple perspectives.

3.1 Conformance Checking Techniques

In this subsection, we describe basic token replay and trace alignment techniques for conformance checking.

Conformance checking has been designed in order to find differences between a process model and a log [4][5]. Most of the techniques can find two types of deviations:

1. Unfitting behavior.
2. Additional behavior.

Unfitting behavior describes a type of deviations where the behavior is observed by a log but disallowed by a process model, while *additional behavior* describes the opposite: behavior that is allowed by a model, but which is not observed in a log.

The simplest way to identify unfitting behavior is *token-based replay* technique. The idea is to replay each trace of the log against the model, represented as a petri net. The model follows the trace by its transitions. When the algorithm cannot go ahead because the transition is not enabled, it adds the missing tokens in order to continue. When the trace is completely replayed, all the left tokens considered as *remaining tokens.* The fitness between a log and a process model is quantified as a number of added and remaining tokens.

The token-based replay approach shows good performance, however, it has some critical weak points. The main one is that it cannot identify the minimum number of errors in respect to log and process model comparison. Trace alignment technique can address this weak point. Each trace in the log is compared to the process model, as a result, we have the closest trace of the process model in respect to the given trace of the log with highlighted activities where the algorithm has found mismatches.

Here, we describe trace alignments, one of the most popular technique to do Conformance Checking.

In trace alignment, we compare every single trace of the event log to the process model. At the end of the trace alignment process, we have an alignment for each trace.

Usually, alignments are represented as a list of skipped and/or inserted activities. Let's describe concepts that are used in trace alignments:

1. *Move on log* indicates that the behavior in the event log cannot be replicated in the process model (BPMN model). So, in order to continue, we move to the next event in the event log.
2. *Move on model* indicates that the expected occurrence of the event in the model doesn't happen in the event log. So, in order to continue, we move to the next activity in the process model.
3. *Synchronous move* tells us that the expected behavior in the log match to the expected behavior in the process model and we move the log and the model to the next step.

In the example of an alignment below (Figure 14) we show an alignment for the Trace 2 from Figure 1 in respect to the process model from Figure 2.

Log	IL	>>	ID	IC	FE	IaD	RaR	IClo
Model	IL	IC	ID	>>	FE	IaD	RaR	IClo

Figure 14: Alignment example

Note: For better visualization we made short names for activities: IL -> Incident logging, IC -> Incident classification, ID -> Initial diagnosis, FE -> Functional escalation, IaD -> Investigation and diagnosis, RaR -> Resolution and recovery, IClo -> Incident closure

The first occurrence of "Incident Classification" is a *move-on-model*, the second occurrence is a *move-on-log*.

In real life processes, some traces fit the process model perfectly. In these cases, we don't even need to compute any alignments and there are no deviations. However, for those traces that don't fit the model we need to run some algorithm that can compute alignments. Most of the trace alignment algorithms use A^* or some modification of it.

A^* (A-star) is a search algorithm. The algorithm is widely used in artificial intelligence in order to do a search in state space. For computing alignments, we need to know what is a state space in our task, initial state, final state and how to generate new states:

1. *State*: A pair of states $< S_1, S_2 >$ such that S_1 is a state of the process model and S_2 is a position in the trace.
2. *State Space*: All possible states as defined above.
3. *Initial state*: Model's start event and the beginning of the trace.
4. *Final state*: Model's end event and the end of the trace.
5. *Generator*: If a synchronous move is impossible do either *move-on-log* or *move-on-model*.

A-star is a heuristic based algorithm, that means that we need to provide some cost function in order to determine optimal alignment. Very basic and most widely cost functions is *the number of move-on-log and move-on-model moves*. You can find an example of the flow of the A-star algorithm below (Trace 2 from Figure 1 in respect to the process model from Figure 2).

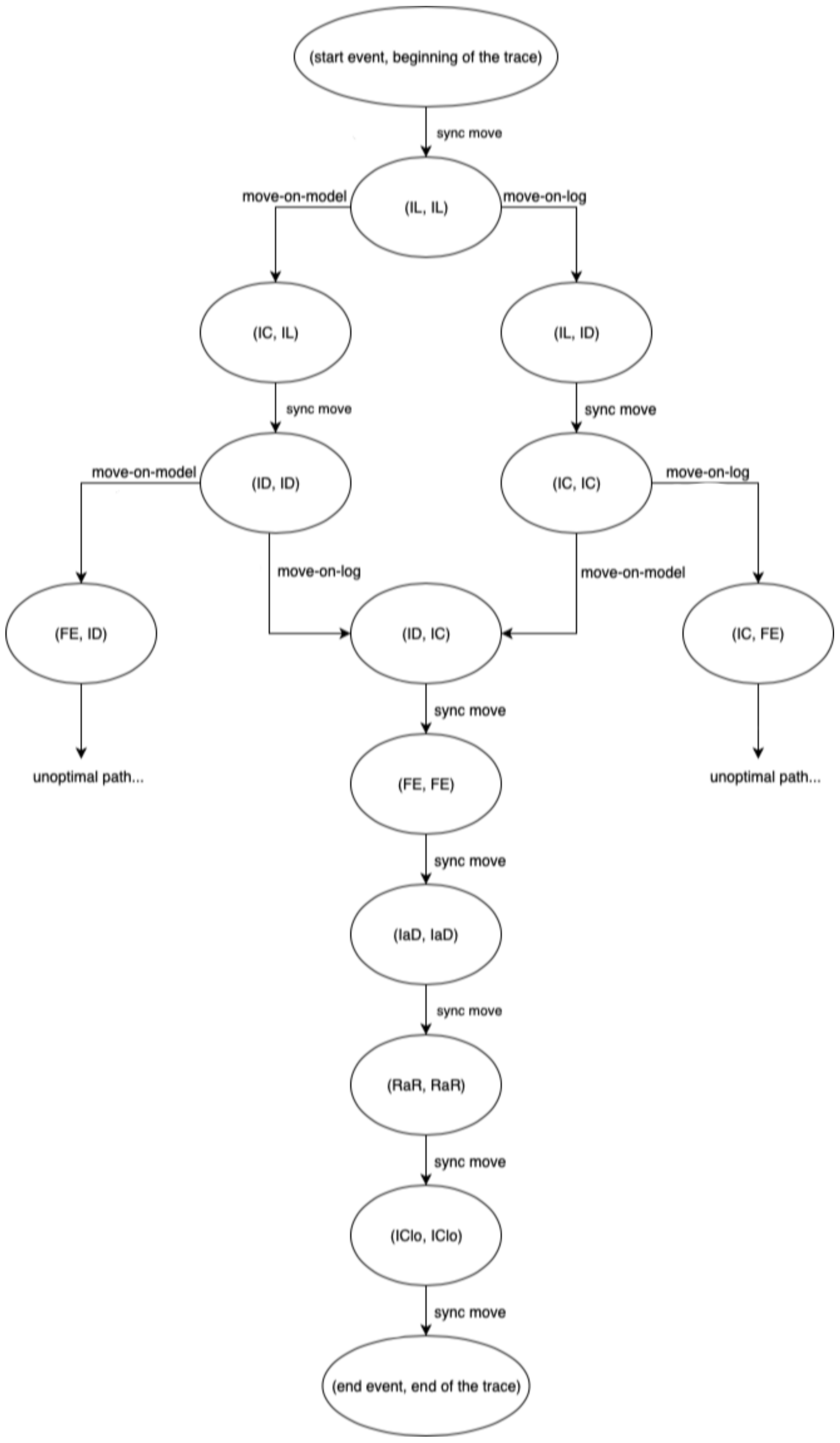


Figure 15: A-star example for computing alignments.

The alignment is represented as a path from the initial state to the final state. As we can see there are 2 optimal alignments for this trace and the model. One of these alignments is illustrated in Figure 14.

The original trace alignment algorithm for conformance checking by Adriansyah [7] assumes that the model is represented as a petri net. Each trace is then represented itself as a (linear) petri net and the alignment is performed over the model's Petri net and the Petri net of the trace. An alternative approach [8] uses automata as an intermediate representation of the control-flow of the process. In this approach, the process model (e.g. BPMN model) is converted into an automaton, the log is also converted into an acyclic automaton and the two automata are then aligned.

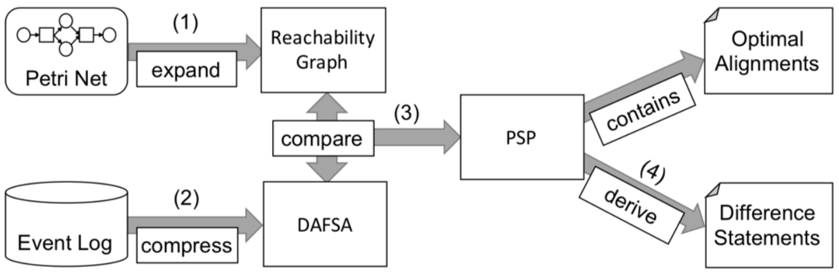


Figure 16: Overview of the automata-based approach

In simple words, an automaton is a graph where vertices represent states of the process and edges represent transitions between states.

When we have the automaton of the process model and the automaton of the event log, we can align them as it was described in the previous subsection.

In this subsection, we described conformance checking approaches. Token-based replay approach shows good performance, but the results are non-optimal. The trace alignment technique has optimal results but the performance of it is not as good as we want.

3.2 Decomposed Conformance Checking

In this subsection, we are going to describe a divide-and-conquer approach that is potentially able to speed up the computation of alignments for the conformance checking.

Most of the conformance checking algorithms have an exponential complexity [1][2][3]. The author from [1] proposes to decompose the original model into smaller parts in order to reduce the computation time. Indeed, if we can divide the model and run computations concurrently, it will speed up computations a lot. However, in this case, we need to have a good algorithm for model decomposition.

Process decomposition is a well-known task and has been described in many papers [1][6]. In [1], the author proposes to use a Single-Entry Single-Exit (SESE) algorithm to decompose process models. The idea of the algorithm is very simple: each decomposed fragment should have one single entry point and one single exit point so that the fragments are completely isolated and independent. You can find the examples of model decomposition taken from [1] in Figure 17 and Figure 18.

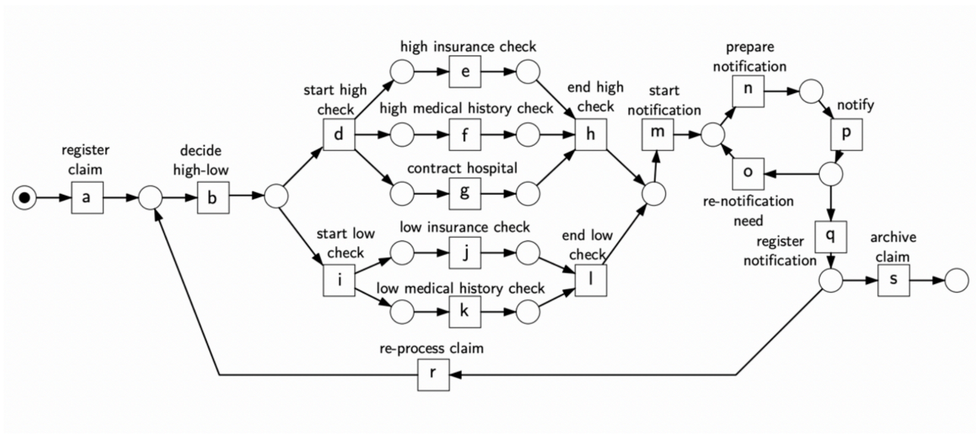


Figure 17: Petri net of claims handling in an insurance company.

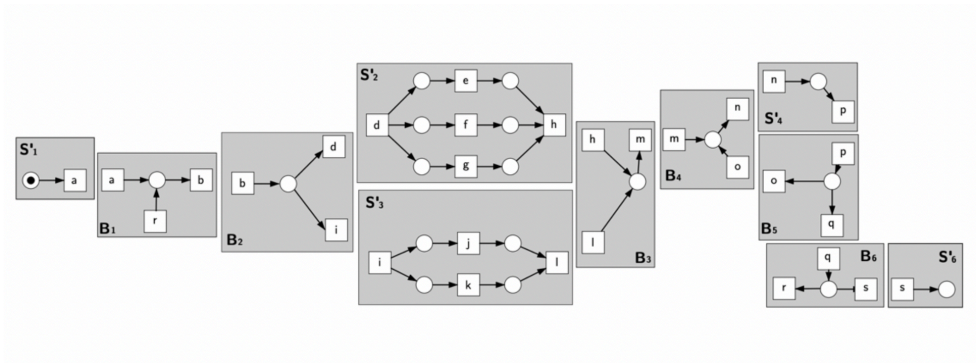


Figure 18: Decomposed petri net of claims handling in an insurance company.

3.3 Multi-perspective Conformance Checking

As it was mentioned before, conformance checking may include several perspectives. In order to get the most accurate results, we need to consider all these perspectives. In this section we describe simple conformance checking approaches to achieve our goal.

The authors from [2] propose an approach in which they put the control-flow perspective on the first place. In other words, they consider the control-flow perspective as more important than the others. According to this method, if the control-flow perspective did not catch a deviation and other perspectives found it, it is possible that the algorithm decides not to consider this deviation. This approach has some problems. The main one is that the solution (the computed alignments) is not always optimal as described. However, this algorithm is quicker than the next one, described below.

As opposed to [2], the authors from [3] propose to consider that all the perspectives have the same level of importance so that the solution gains optimality. Optimality, in this case, is defined according to the cost function defined in [3]. However, this approach is very computation consuming and when we have terabytes of the data, this factor is very important.

So, as we can see, the approaches described in this subsection have clear limitations. However, these methods can be used if optimality is not required or when the amount of data is small enough. However, by combining the divide-and-conquer approach and multi-perspective conformance checking techniques we potentially can get optimal results in a reasonable time. In [1], the author extends the decomposed conformance checking algorithm, so that it fits the data perspective as well. The extension is very simple: a variable must not belong to more than one decomposed fragment. In other words, any two decomposed fragments cannot write or read the value from the same variable. And it makes sense, if two model fragments can read/write from the same variable, they are not independent anymore, which means that the decomposition is not valid.

4 Approach

In this section, we introduce a chosen approach in order to do conformance checking with data perspective. We will describe challenges and implementation features in respect to Elixir (chosen programming language).

As a conformance checking technique, we chose automata-based conformance checking [8]. According to this technique, we have to transform the process model to the automaton. Let us consider the following process model taken from Figure 2 but with numbered sequence flows (Figure 19).

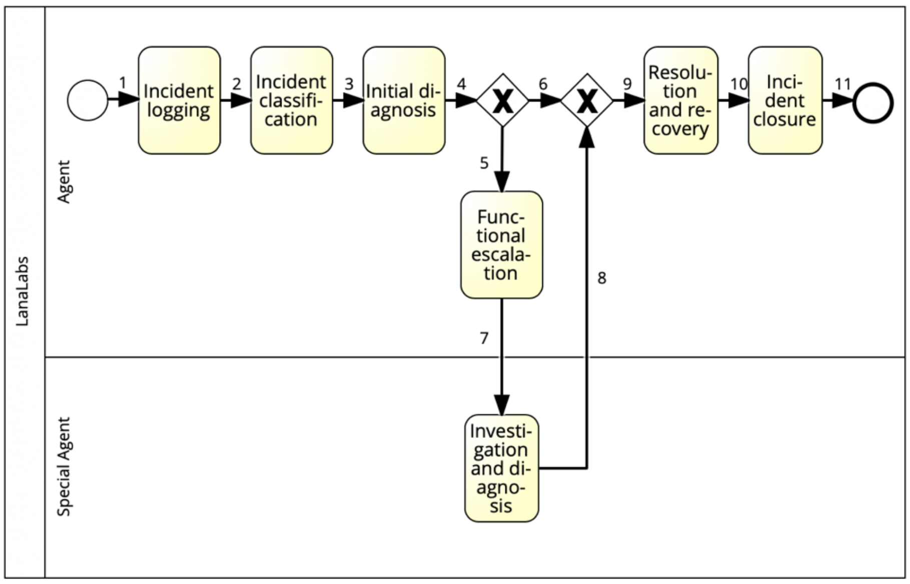


Figure 19: Incident Management process model with numbered sequence flows.

Automaton is a state machine. So, in order to build an automaton for the process model above we need to define a state and transitions between states:

- *State*: A set of active sequence flows.
- *Transition*: An Activity between sequence flows.

Having these definitions, we can build an automaton for the process model from Figure 19.

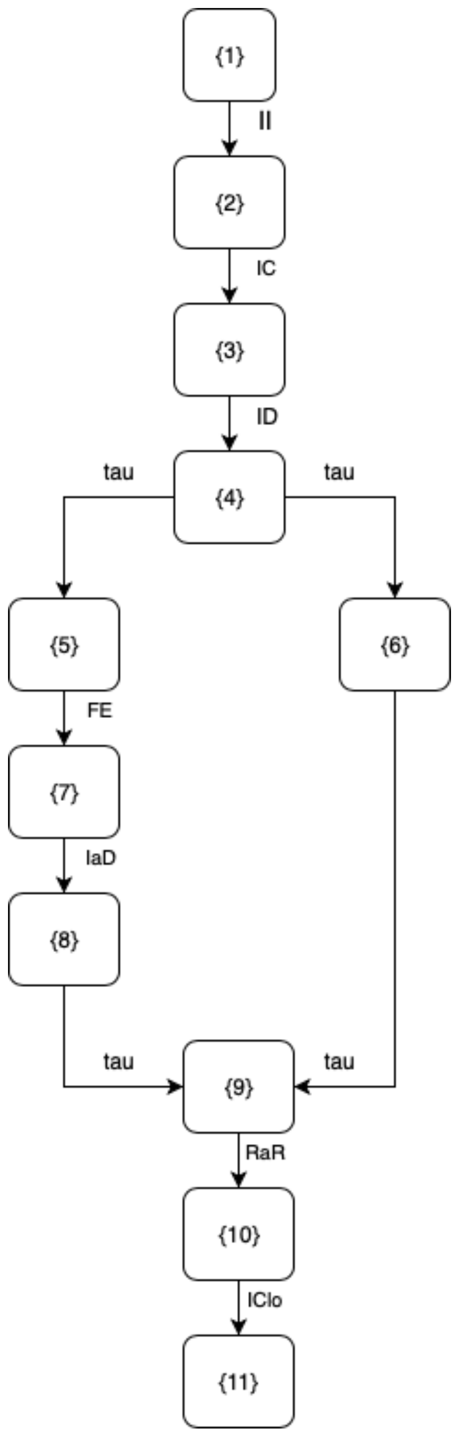


Figure 20: Incident Management automaton.

For the automaton, we need to remove tau-transitions. Tau transitions represent invisible actions that are not recorded in the event log. We are interested only in visible (activity) transitions because only those make sense for us. That's is why it is good to remove tau transitions from the automaton. In this thesis, we propose to remove tau transitions by combining the states connected by the tau transition so that a new state has all incoming sequence flows of the source state of the tau transition and it has all outgoing sequence flows of the target state of the tau transition. The new state itself represents all the sequence flows from both of the states.

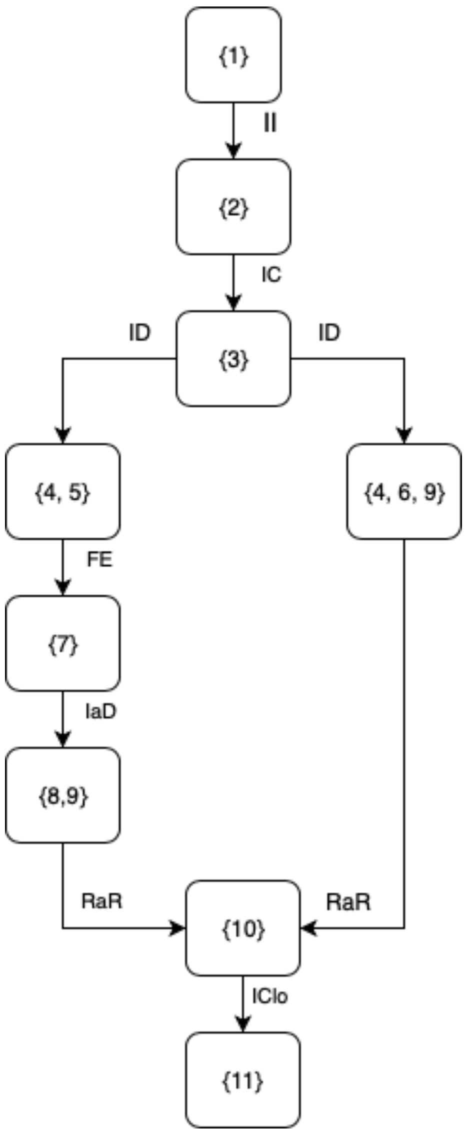


Figure 21: Tau-less Incident Management automaton.

In Figure 21 we can see the finale automaton without tau transitions for the process model from Figure 19.

Let’s now consider data-aware conformance checking. In Section 2 we have changed a process model from Figure 2 by adding a condition to the sequence flow (Figure 10). In order to build an automaton for this process model, let’s assume that it has the same numbered sequence flows as on Figure 19.

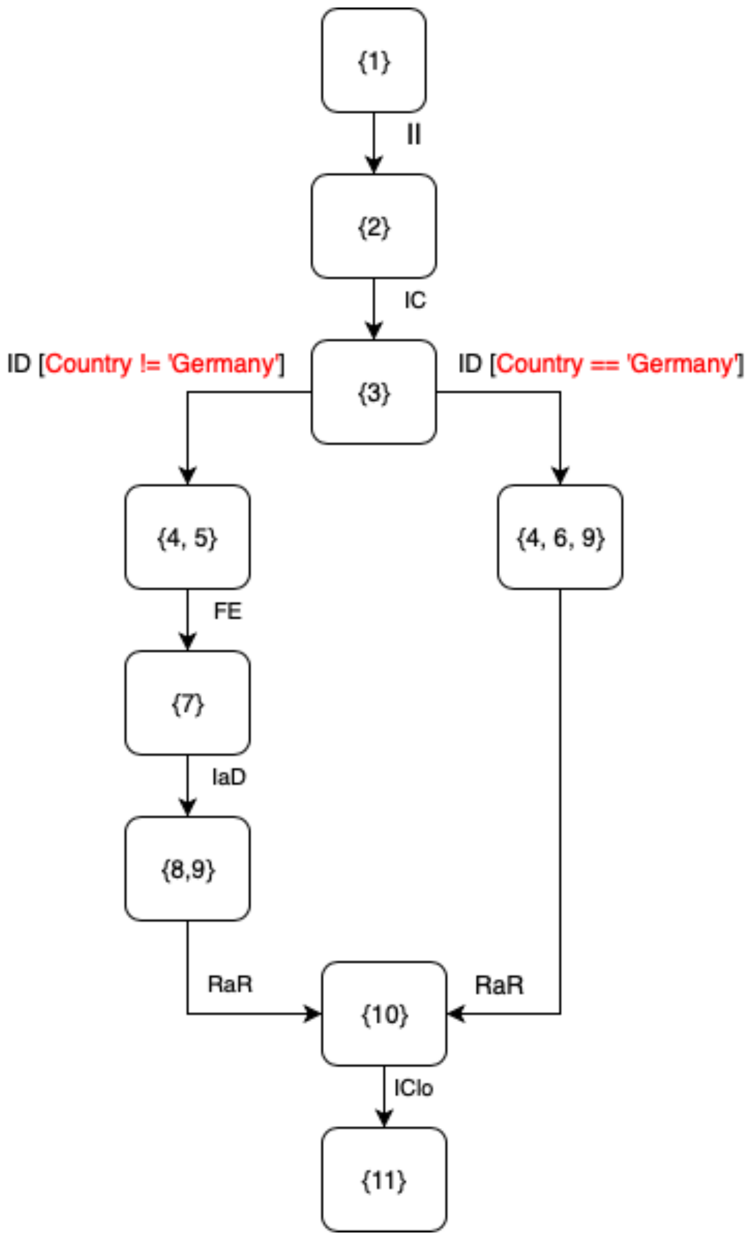


Figure 22: Incident Management automaton with conditions.

As we can see from Figure 22, there is a condition for “Initial diagnosis” flow. It means that in order to move on through this flow, the trace must have a correct next activity (Initial diagnosis) and the correspondent condition must be true.

It could happen that we have a state with multiple active sequence flows (*{4, 6, 9}*, for example). In this case, we have to combine conditions of these sequence flows by *AND* operator.

We assume that by default all sequence flows has no conditions, it also can be represented as a sequence flow with a condition with *true* value so that we don’t differentiate sequence flows with and without conditions. It makes building an automaton easier.

Having an automaton of the process model we can already align it to the traces. In Section 3 we described an automata-based technique to do conformance checking. According to this technique, we need to compress an event log to DAFSA (Deterministic Acyclic Finite State Automaton). In this thesis, we do not do this but just align each unique trace to the process model automaton. It means that we lose some performance because if some traces have the same prefix, we do the same job by aligning this prefix multiple times. But if we have compressed event log to DAFSA, we align this prefix only once so that we save some time and we have better performance.

Compressing an event log to DAFSA can be a direction for the future work of this thesis.

The alignment algorithm should be slightly changed in order to fit the data perspective. As it was described in Section 3, we use A^* as a base algorithm for computing alignments. In control-flow conformance checking, if the synchronous move is possible, there is no other way but to do it. In data-aware conformance checking, synchronous move is not always an optimal step, that’s why we need to consider all possible steps to be performed. So, every time when we need to expand the state, we generate new (next) states by applying all possible synchronous moves, move-on-logs and move-on-models and then chose the best way according to the cost function of the algorithm as it was described in Section 3.

As an example of data-aware alignment, let’s align Trace 1 from Figure 1 with trace attributes from Figure 11 to the process model automaton from Figure 22. As was mentioned in Section 2, this trace perfectly fit from control-flow perspective, but it has a deviation in respect to data perspective.

Log	IL	IC	ID [Country == 'Germany']	FE	IaD	RaR	IClo
Model	IL	IC	>>	FE	IaD	RaR	IClo

Figure 23: Example of data-aware alignment.

5 Implementation and Evaluation

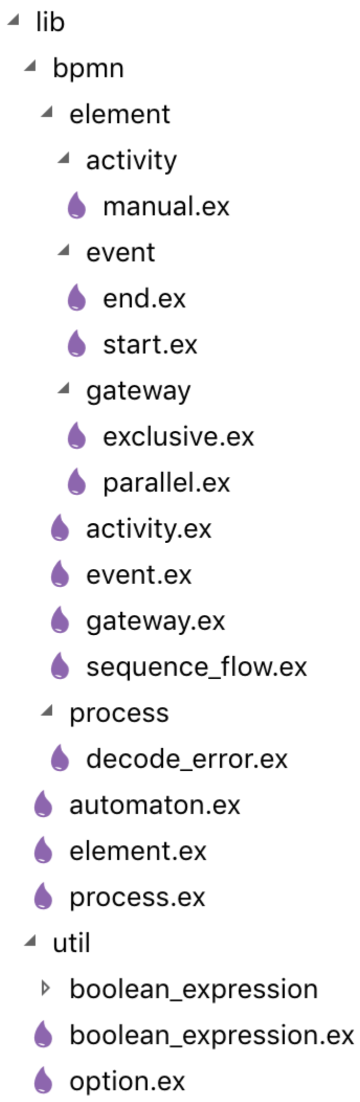
In this section, we describe implementation features, structure of the code of the project and evaluation of the written algorithms.

5.1 Implementation

Elixir has been chosen as an implementation programming language for this thesis. Elixir language is based on the *Erlang* programming language, which means it shares the same abstractions for building applications. Elixir is a functional language. The functional approach is very popular nowadays. It allows to write and run concurrent distributed applications out of the box. In the era of big data, distributed applications it is rather a need than an option, that is why, I think, Elixir is very popular.

Turning to the process mining, event logs becoming very huge, from gigabytes to terabytes. In order to analyze this amount of data, we need to do it concurrently. Elixir helps us a lot in it because it provides a lot of tools for writing concurrent applications.

Elixir has a lot of libraries for different needs. However, there is no BPMN library as well as any library that is able to do conformance checking. So, another purpose of this thesis is to provide a BPMN library with conformance checking algorithms for Elixir community.



```
lib
  bpmn
    element
      activity
        manual.ex
      event
        end.ex
        start.ex
      gateway
        exclusive.ex
        parallel.ex
      activity.ex
      event.ex
      gateway.ex
      sequence_flow.ex
    process
      decode_error.ex
    automaton.ex
    element.ex
    process.ex
  util
    boolean_expression
    boolean_expression.ex
    option.ex
```

Figure 24: Code structure

In Figure 24 we show how the structure of the project looks like. Elixir community has code guidelines, so that the opensource community can easily contribute to the projects. These guidelines are basically about the code and project structure. For example, interfaces (base classes) should be places in the same level as the folder that include classes that implement this interface. Meanwhile, this folder should have the same name as the interface itself. We can clearly saw this in Figure 24. During the development process, we followed these guidelines.

BPMN format itself is very huge and contains a lot of elements. In this thesis, we designed a library that includes only the core BPMN elements. Specifically, we selected a subset of BPMN elements known as process graph [9] with addition of start and end events. Other elements could be added later as a future work for this thesis.

In order to parse and process sequence flow conditions, we need to have some boolean expression parser. Unfortunately, there was no such a library for Elixir, so we designed and implemented this parser from scratch on Elixir.

Most of the code is opensource and available on *Github* (https://github.com/imaxmelnyk/rainbow). However, the alignment algorithm is taken from *LanaLabs* and is not present in the opensource project.

5.2 Evaluation

For the experiments we use demo event logs and process models provided by *LanaLabs*. The provided data has a different number of activities, traces, event etc. We can see the datasets statistics in Figure 25.

Dataset	**Events**	**Unique Events**	**Traces**	**Unique Traces**	**Model size**
Incident Management	11674	7	2000	8	22
Sales Process	4712	15	572	156	38
Manufacturing Dataset	241280	65	94612	759	98

Figure 25: Experiment datasets statistics.

Note: Model size is a number of nodes (activities, events, gateways) and sequence flows in a process model.

As we can see from Figure 25, the datasets are very different. “Incident Management” dataset is the one we used in this thesis as an example. It is the simplest and the smallest dataset. “Sales Process” dataset is more complicated, it has less events and traces, but it has more unique traces (variants) and more complicated process model. The most complicated dataset is “Manufacturing Dataset”. It is very close to real-life story. We can see the huge number of unique traces and events. It has a very complicated process model which is not even readable without proper filtering.

As evaluation metrics we use:

1. *Log preprocessing time*. This metric gives as an idea of how complicated the event log itself and how much time the program needs to parse the log and compute unique traces.
2. *Alignment computation time*. This metric is the one we need in order to evaluate the alignment algorithm itself.

Dataset	**Preprocessing time (ms)**	**Alignment computation time (ms)**	**Total (sec)**
Incident Management	379	539	0.91
Sales Process	479	1321	1.8
Manufacturing Dataset	6023	9447	15.47

Figure 26: Evaluation metrics.

As we can see in Figure 26, for the simple logs the preprocessing and alignment computation is fast enough. And for the real-life log the total time is significantly longer. This is something we expected as a result.

For the current version of the algorithm, it doesn’t matter, if sequence flows have conditions or they don’t, the alignments computation time remain more-less the same. This is because the conditions are checked only once for each trace (preprocessing step) and it is a simple task. The main complexity is in A^* search. As it is described Section 4, synchronous moves are no longer the only possible way if they exist. We expand states for every possible move. It means that the algorithm must check significantly more states and alignments. It is, however, possible to choose how to expand states. If the data perspective is not present, we expand states as it was described Section 3, and if the data perspective is present, we run the algorithm from Section 4. This can be implemented as a future work of this thesis.

In this section, we explained how the algorithm described in Section 4 has been implemented and what tools we used in order to do that. We showed the performance statistics of the implemented algorithm in respect to different datasets. There are, however, some improvements could be done in the future.

6 Conclusion

Process mining is very popular nowadays due to complex processes and large amount of data. Conformance Checking is one of the process mining tasks that helps companies to check their real process executions in respect to designed process models and catch deviations if there are any.

However, most of the conformance checking papers are focused only on the control-flow perspective omitting the data perspective. In this thesis we designed and implemented the algorithm that can deal not only with control-flow perspective but with data perspective as well.

In this thesis we consider data perspective as boolean conditions on sequence flows, where variables are taken from trace attributes of the event log. We have chosen the automata-based technique for conformance checking as a base for our work. We slightly changed alignment algorithm so that it fit the data perspective. At the end we were able to compute optimal data-aware alignments.

For the implementation we choose *Elixir* programming language. *Elixir* is popular due to its functional approach but there is no BPMN library, neither Conformance Checking library. That is why we decided to contribute to development of this programming language by implementing those libraries.

For evaluation we used the datasets provided by *LanaLabs*. The results of the evaluation weren't a surprise. It is more-less the same with data perspective and without it. That's all because of modified (data-aware) alignment algorithm that needs to generate significantly more states in order to compute optimal alignments.

A lot of work is done; however, a lot still can be done. We highlight 5 main points for the future work of this thesis:

1. Extending the BPMN library with all the elements from BPMN format. For now, the library includes only core elements.
2. Converting event logs to DAFSA (Deterministic Acyclic Finite State Automaton) so that the same prefixes of different traces aren't aligned twice. For now, the algorithm aligns all unique traces to the automaton.

3. Modify the alignment algorithm, so that it can identify whether or not the data perspective is present in the model and will run different algorithms accordingly. This should make the tool faster when there is no data perspective.
4. The developed tool only takes into account case attributes. One possible extension is to handle event attributes as well. This might be challenging since the value of event attributes can change during a trace and this has an impact on the performance of the algorithm.
5. The experimental evaluation could be extended in order to examine the tradeoffs between the proposed approach and alternative ones such as the one in [3].

References

[1] Jorge Munoz-Gama. Conformance Checking and Diagnosis in Process Mining - Comparing Observed and Modeled Processes. Lecture Notes in Business Information Processing 270, Springer 2016, ISBN 978-3-319-49450-0, pp. 1-195

[2] Massimiliano de Leoni, Wil M. P. van der Aalst. Aligning Event Logs and Process Models for Multi-perspective Conformance Checking: An Approach Based on Integer Linear Programming. BPM 2013: 113-129

[3] Felix Mannhardt, Massimiliano de Leoni, Hajo A. Reijers, Wil M. P. van der Aalst. Balanced multi-perspective checking of process conformance. Computing 98(4): 407-437 (2016)

[4] Luciano García-Bañuelos, Nick van Beest, Marlon Dumas, Marcello La Rosa, Willem Mertens. Complete and Interpretable Conformance Checking of Business Processes. IEEE Trans. Software Eng. 44(3): 262-290 (2018)

[5] Wil M. P. van der Aalst, Arya Adriansyah, Boudewijn F. van Dongen. Replaying history on process models for conformance checking and performance analysis. Wiley Interdiscip. Rev. Data Min. Knowl. Discov. 2(2): 182-192 (2012)

[6] Massimiliano de Leoni, Jorge Munoz-Gama, Josep Carmona, Wil M. P. van der Aalst. Decomposing Alignment-Based Conformance Checking of Data-Aware Process Models. OTM Conferences 2014: 3-20

[7] Arya Adriansyah, Boudewijn F. van Dongen, Wil M. P. van der Aalst. Conformance Checking Using Cost-Based Fitness Analysis. EDOC 2011: 55-64

[8] Daniel Reißner, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, Abel Armas-Cervantes. Scalable Conformance Checking of Business Processes. OTM Conferences (1) 2017: 607-627

[9] Artem Polyvyanyy, Luciano García-Bañuelos, Marlon Dumas. Structuring acyclic process models. Inf. Syst. 37(6): 518-538 (2012)

Licence