Master in Computing
Llenguatges i Sistemes Informàtics

Master Thesis
– September 2010 –

ALGORITHMS FOR PROCESS CONFORMANCE AND PROCESS REFINEMENT

Student:	*Jorge Muñoz-Gama*
Advisor/Director:	*Josep Carmona Vargas*

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Title: Algorithms for Process Conformance and Process Refinement

Author: Jorge Muñoz-Gama

Advisor/Director: Josep Carmona Vargas

Date: September 2010

Abstract: Process Conformance is a crucial step in the area of Process Mining: the adequacy of a model derived from applying a discovery algorithm to a log must be certified before making further decisions that affect the system under consideration.

In the first part of this thesis, among the different conformance dimensions, we propose a novel measure for precision, based on the simple idea of counting these situations were the model deviates from the log. Moreover, a log-based traversal of the model that avoids inspecting its whole behavior is presented. Experimental results show a significant improvement when compared to current approaches for the same task. Finally, the detection of the shortest traces in the model that lead to discrepancies is presented.

In the second part of the thesis, two different approaches are proposed in order to use the precision analysis information for refining the model, improving its accuracy. The first one is based on the idea of break concurrencies reflected in the model but not in the log. The second one presents the Supervisory Control Theory as the mechanism to improve the accuracy of the models building supervisors for controlling the precision issues.

Keywords: Process Mining, Process Conformance, Process Refinement, Petri Nets, Supervisory Control

Language: English

Modality: Research Work

"Audentes fortuna iuvat"
(Fortune favours the brave)

Virgil's Aeneid

Acknowledgments

Thanks to my advisor Dr. Josep Carmona, for his tutelage, his guidance and his patience, providing me the opportunity to dive into the research world.

Special thanks to Marc Solé, Anne Rozinat, David Aguilera and ProM developers for their help.

Finally, I would like to thank my family for being always behind me in the good moments, but also in the bad ones.

This work has been partly supported by the *Spanish Ministry of Education and Science* under FORMALISM project (TIN2007-66523).

This work has been partly supported by a grant of *Intel Corporation*.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

1.1 An Overview on the Conformance and the Refinement Problem

International enterprise, hospitals, universities, on-line shops, small companies, . . . , all them involve a huge number of people performing a huge number of processes. These processes have become more and more complex, and they require a more rigid and close control to avoid errors. But at the same time they need more flexibility to deal with unexpected situations. For that reason, during the last decade, many Workflow management systems, such as Staffware or IBM MQSeries, have appear in order to support business processes. Even other business software, such as SAP or PeopleSoft, has adopted workflow technology [15].

This workflow technologies require the use of process *models*, i.e., more or less restrictive representations of the process, describing its flow, and the possible paths it can take. How to obtain these models is the main handicap of workflow systems. One possibility is to create them *manually*. It is a cost-consuming task that require time and the need of experts in the domain of the process and the modeling language. However, in the last years, more and more techniques have come up to build models *automatically*. These techniques, called *Process Mining discovery* approaches, aim to mine a model from the process executions dumped in the system logs. These logs contain annotations of the transactions and the events produced during the execution of the processes [46].

Once the model has been obtained (manually or using some discovery technique), a new requirement comes up: measuring the quality of the model. In other words, quantify the quality of the model to describe the real processes whose executions are reflected in the logs. How good is the model describing the process? How much of the process is covered by the model? Is the model informative enough? Is the model precise enough or it includes more information than it should do? Is the model general enough to include changes in the process, or it is too much tied with respect to the examples used to create it? Is the model structure clear, compact and meaningful, or it contains elements that inflate it and make the model less understandable? These and other questions are the ones addressed by the *Process Conformance* analysis techniques [29].

The use of workflow management concepts and technologies may produce great benefits for the companies, as long as the models used are correct, i.e., they describe accurately the processes of the company. However, the use of erroneous models could produce catastrophic consequences. Even if the model was correct time ago, the processes may change and evolve, turning on the model into an out-of-date reflex of the reality. In such cases, one option is to get rid of the old model and build a new one from scratch. This task could suppose a huge cost in terms of money, but also in time. Even using some process discovery algorithm, the results may not be adequate. On the other hand, an other option could be to use the existing model, and *correct* it. This is known as *Process Refinement.* Refining a model, we obtain a more accurate model by correcting its errors, or updating it to reflect precisely the current processes.

1.2 Aim of this Work

The goals of this thesis are addressed in two directions: Process Conformance and Process Refinement.

Concerning about *Process Conformance*, the aim of this thesis is to propose a new approach to measure the *conformance* between a model and a log. In our case, the modeling language chosen are the *Petri Nets.* Among all the aspects conforming the conformance analysis, the technique proposed will focus on *Precision*, i.e., measure how accurately the model describes the process, avoiding include more behavior than the one reflected in the log. This new measure will be based on the effort needed to achieve a completely precise model, as estimator of the precision. Our approach must be an alternative to solve the efficiency problems related with the current approaches for similar scenarios, making our approach efficient enough for real benchmarks. Together with the measure of precision, our approach must provide mechanism to *locate* where the precision issues are, identifying effectively the possible points of improvement. Finally, the approach must be implemented in a tool, making possible to tests the results of the conformance analysis.

Concerning about *Process Refinement*, the aim of this thesis is to introduce the results of the precision analysis as part of the refinement procedure. In particular, two techniques are proposed to refine the precision of a given model (a Petri Net in our case). The first algorithm proposed, called *Breaking Concurrencies*, is based on the idea of remove concurrencies of the model not reflected in the log, in order to improve the precision. The second proposal of this thesis is the use of *Supervisory Control* as a mechanism for precision refinement.

1.3 Organization of the Thesis

This work is organized in chapters as follows:

- In Chapter 2 we introduce the concept of Process Mining, the area where this approach belongs to, presenting the different perspectives and the existent types of Process Mining, including Conformance and Refinement. The chapter reviews the state-of-the-art of the area and the related work previously done. Finally, the chapter includes a deeper review on the Conformance process, and the aspects composing it.

- In Chapter 3 we introduce the two main elements involved in any Process Mining technique, the log and the model, and the possible association between them. This chapter also includes a deeper description of the the modeling language chosen for our approach: Petri nets. Finally, we present Transition Systems as a needed element to understand the techniques proposed in this thesis.

- In Chapter 4 we present our approach for Process Conformance. The first sections are addressed to the performance of the procedure, while the next ones present the resulting elements of the conformance analysis: the metric (to evaluate the conformance degree) and the set of Minimal Disconformant Traces (to locate the possible conformance problems). Finally, in the last section of the chapter we present some extensions and variants of the main approach, in order to deal with special cases.

- In Chapter 5 we present the implementation of the conformance approach proposed in this thesis. The first section introduces ProM, the framework of Process Mining used to implement our approach. In the last section we present ETConformance, the ProM plug-in that implements the approach.
- In Chapter 6 we detail the experimental results obtained after using our tool in a benchmarks set.
- In Chapter 7 we present our approaches for Process Refinement. The first section addresses the Breaking Concurrencies technique, while the second section propose the use of the Supervisory Control Theory as base for future refinement techniques.
- In Chapter 8 we illustrate the approaches presented in this thesis presenting a complete case of study.
- Finally, in Chapter 9 we elaborate the conclusions of this thesis and we suggest ideas for future work.

2 Process Mining, Process Conformance and Process Refinement

In this chapter we present *Process Mining*, i.e., the process management techniques that allow the analysis of business processes based on logs of their executions. This presentation includes a complete review of the previous work done in the area, and it details the different perspectives that could be analyzed using Process Mining techniques (cf. Sec. 2.1.1). In addition, we present the classification of the Process Mining approaches based on the existence of a-priori model of the process, and how this is used (cf. Sec. 2.1.2). Among others, this classification includes the categories of Process Refinement and Process Conformance, final goals of the work presented in this thesis. The last one, Process Conformance, is presented in detail in Sec. 2.2, including the four dimensions composing the conformance analysis.

2.1 Process Mining

The world is composed by processes. The examinations or treatments a patient undergoes in a hospital, or the purchase of some product in a online shop are examples of everyday processes. Hospitals and shops, international enterprises and small companies, universities and schools, ... all them involve people performing processes. For most of them, it is necessary the use of some information system product in order to support and control this huge number of processes.

There are a wide variety of different products to support processes [15], such as the *Workflow Management Systems*, systems that allow you to define and control the flow of a process. Recently, other business software, such as *Enterprise Resource Planning* (ERP) or *Customer Relationship Management* (CRM), has adopted workflow technology too. Even *Web Services* or *Software Configuration Management* tools (SCM), can be seen as process-aware systems. But all these different information systems has one thing in common: they record logs.

The logs recorded by the information systems contains annotations of the events, transactions or exceptions produced during the execution of the processes. In that sense, logs are the closest reflect of the real processes, i.e., they are not biased by any preconceived idea about the process, but they only reflect accurately the reality.

The idea of Process Mining techniques are to use this unbiased reflect of the reality (the logs) to derive models describing the processes [45, 3]. These models can be used to analyze the processes, detecting possible problems and bottlenecks, or to check how some change would affect the performance of a process. But these models can be also used for tune, configure or redesign the information systems used to manage the processes. Although the derivation of a model from a log is the most common scenario, the Process Mining techniques can be also applied for checking the correctness of an a-priori model with respect to the reality, reflected in the log[1] [29].

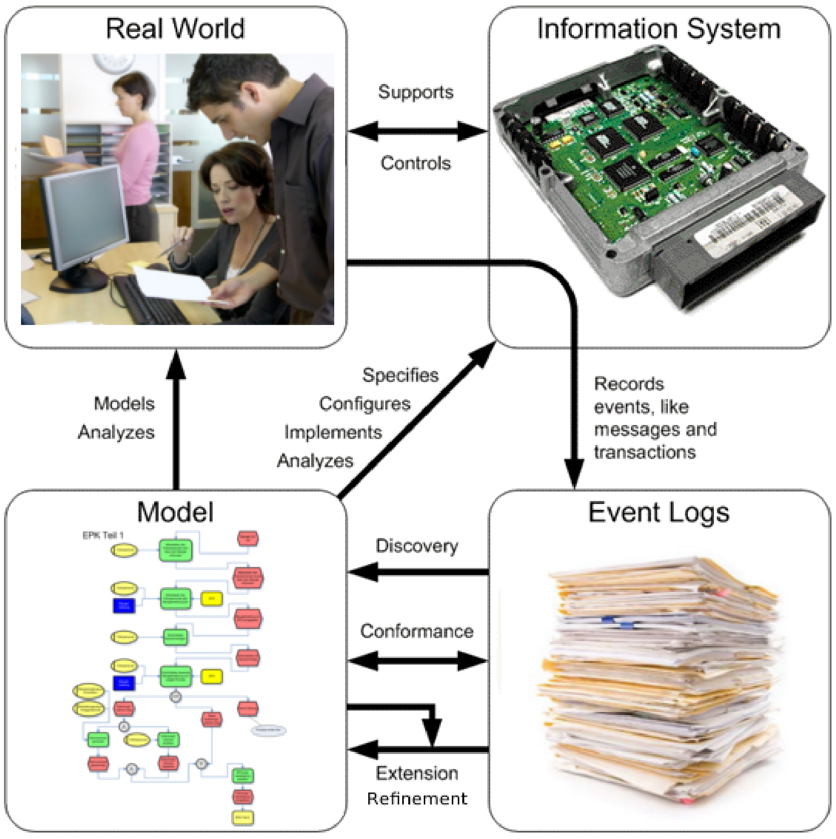


Figure 2.1: *Diagram of Process Mining. This figure is based on one that can be found in [3].*

2.1.1 Perspectives of Process Mining

The main goal of Process Mining is to obtain some information about a process from its track in a log. However, the kind of information wanted may be different in each particular case. This is because a process is a complex entity, composed by several aspects called *perspectives*. In Process Mining we distinguish three perspectives of a process: *process perspective*, *organizational perspective* and *case perspective*.

- ***Process Perspective***
  The process perspective focuses on the *control-flow* of the process, i.e., it describes the sequence of events of the process, how they follow to each others, and the ordering relation between them. There are several languages for modeling a process control-flow, each one with its own characteristics that make it more appropriate in some cases than other

[1]The different classes of Process Mining techniques (Discovery, Conformance, Extension and Refinement) are discussed in Sec. 2.1.2

languages. Some examples of flow modeling languages are the Petri nets (formal and with a solid research base behind), or the Event-driven Process Chains (intuitive and easy to understand for non experts). Both languages will be presented in detail in the following chapters of this thesis. Fig. 2.2 shows an example of control-flow modeled using Petri nets.

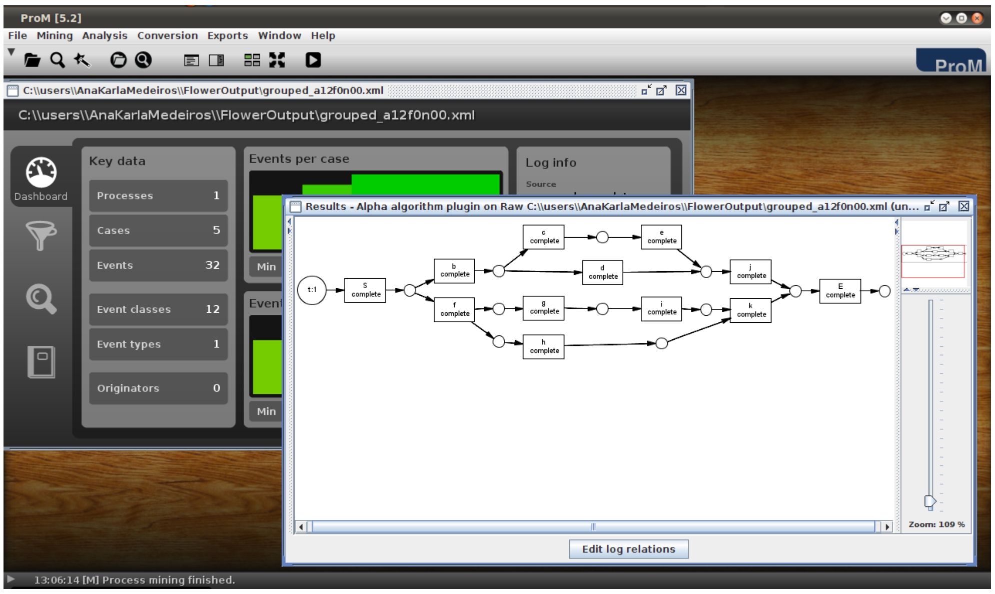
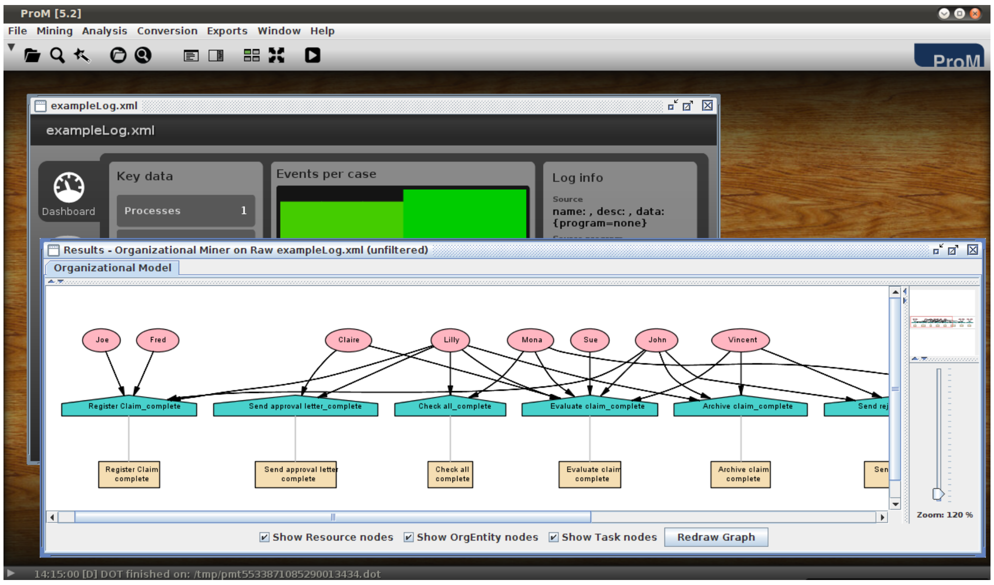


Figure 2.2: *Example of Process Perspective. The snapshot shows a Petri net defining the control-flow of a process. This Petri net has been generated by the $\alpha - algorithm$.*

Most of the algorithms and techniques of Process Mining aim to extract information about this perspective. An example could be the $\alpha - algorithm$ [46] (and the rest of the α-series algorithms [13, 55, 53, 54, 54, 23]), the classical Process Mining control-flow algorithm that aims to extract a process flow model from the log and represent it in terms of a Petri net. Some other examples of algorithms could be the *Genetic Miner* [14] (Process Mining using Genetic Algorithms), the *Heuristic Miner* [51, 52] (a heuristics-driven process mining algorithm), *Genet* [7] and *RBMiner* [39] (to mine Petri nets from Transition Systems), or the *ILPMiner* (also known as *Parikh Miner*) [47] (that uses Integer Linear Programming to extract a model). Process Perspective is also the final goal of this thesis, and all the approaches presented will be focused on the control-flow of a process instead of other perspectives.

- ***Organizational Perspective***
  In organizational case we are not interested in the order of the events, but in the actors involved in the process. The actor of a process could be the person who has executed an action (i.e., the originator), the person who has received the results of the action, or any other people involved in some way. All this information about the actors is annotated in the log, associating it to each event occurred during the execution of the process. The algorithms focused on the Organizational Perspective use the actors data in order to extract and analyze information about the social relations between the performers (i.e. Social Network), or to group and classify people performing similar tasks in terms of roles and organizational units. An example of algorithm focused on the organizational

perspective is the *Social Network Miner* [43], where the authors present metrics and a tool to extract a social network from a log. In [24] these techniques are applied in order to gain knowledge about the relations between hospital departments in a medical real scenario. Some other examples of methods for the organizational perspective are the *Organizational Miner*, the *Role Hierarchy Miner* and the *Semantic Organizational Miner*, all them implemented as plug-ins of the ProM framework [4]. Fig. 2.3 shows an example of organizational perspective analysis.



14:15:00 [D] DOT finished on: /tmp/pmt5533871085290013434.dot

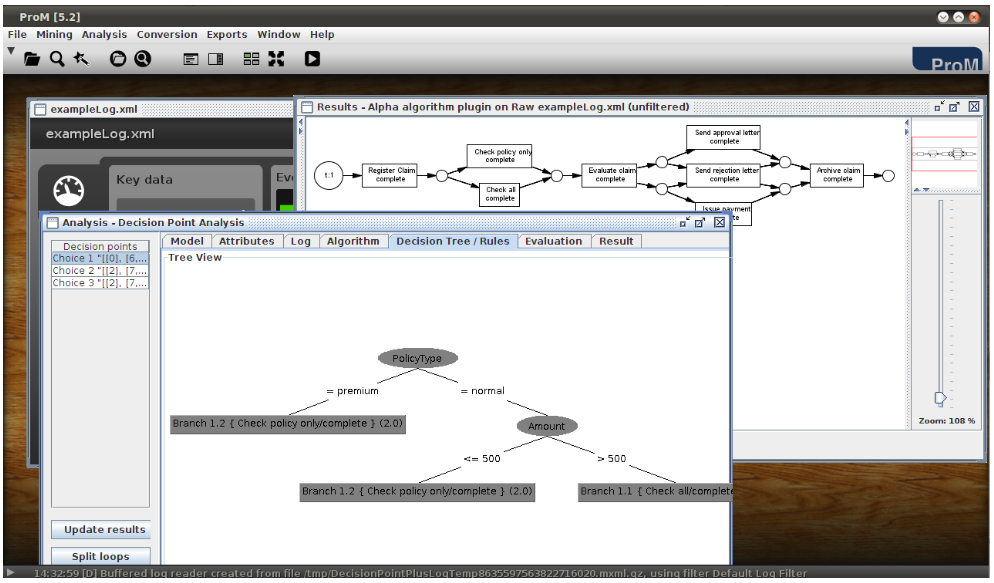
Figure 2.3: *Example of Organizational Perspective. The snapshot shows the results of the organizational analysis performed by Organizational Miner.*

- ***Case Perspective***
  Sometimes the aim of Process Mining is not to extract information about the control flow or the actors, but to extract some other specific aspects or properties about the analyzed process. This is known as the Case Perspective of a process (or it is also called Data Perspective sometimes). There is a wide range of diverse properties in a process that can be analyzed, e.g., information about the maximum, minimum and average time of the actions, or the detection of possible bottlenecks of the process. Most of the algorithms focus on the case perspective take use of the additional data fields[2] associated to the events in the log, in order to perform its analysis (e.g., the timestamp associated to each event log). Some examples of Process Mining methods for the Case Perspective are the *Decision Miner* [32] (that analyze how data attributes influence the choices made in the process), or the *Performance Sequence Diagram*, both implemented in ProM [4]. Fig. 2.4 shows an example of case perspective analysis using the Decision Miner.

2.1.2 Classes of Process Mining

All the techniques in the Process Mining area try to extract information about the real world analyzing the process annotations in a log. However, depending on this *extraction* of information, its final goal and the elements involved, we can differentiate between *classes* of Process

[2]The event log structure and its fields are explained in detail in Sec. 3.1



14:32:59 [D] Buffered log reader created from file /tmp/DecisionPointPlusLogTemp8635597563822716020.mxml.gz, using filter Default Log Filter

Figure 2.4: *Example of Case Perspective. The snapshot corresponds to the analysis performed by the Decision Miner over a Petri net and its event log.*

Mining. There are four classes of Process Mining: *Discovery*, *Conformance*, *Refinement* and *Extension*.

- ***Discovery***
  In this scenario there is a log, reflex of a system, but there is no a-priori model. The aim of the techniques is to *discover* a model that describes the reality.

  This class of Process Mining is the oldest one, and is the class with more algorithms, techniques and research behind. Actually, in some cases the terms *mining* and *discovery* could be considered synonyms, making common the use of Process Mining name to refer the specific Process Mining discovery class. It exists a wide number of discovery techniques in Process Mining, with a wide variety of methodologies and technologies to perform the extraction of the model. Some examples of discovery algorithms are: the α-*algorithm* [46] (which derive a Petri net describing the behavior observed in the log), *Petrify* [10], *Genet* [7] and *RBMiner* [39], (using the *theory of regions* to derive a Petri net), *Genetic Miner* [42] (using Genetic Algorithms), *ILPMiner* [47] (based on Integer Linear Programming), *Heuristics Miner* [52] and *Fuzzy Miner* [19] (heuristics driven process mining algorithms), *Multi-phase Miner* [50] (for deriving Event-driven Process Chains), or the *FSM Miner* [44] (to extract a Transition System from an event log). All the methods above focus on the control-flow perspective of the process. However, as it has been seen in Sec. 2.1.1, a process is composed by some other perspectives. Examples of discovery algorithms for these other perspectives are the *Social Network Miner* [43] or the *Organizational Miner* (both for the organizational perspective).

- ***Conformance***
  Contrary to the Discovery scenario, in the Conformance case it exists a-priori model. The goal in this case is to check if the reality conforms the model. Conformance analysis is used to detect, locate and explain discrepancies between the model and the log, and measuring their severity.

One of the most important works on Process Mining Conformance is [33], where the authors present several metrics to have a complete vision of the conformance between the reality (reflected in a log) and the model (a Petri net in this case). Notice that one of the main parts of this thesis, Chapter 4, is focus on Conformance, where we present a new approach for conformance checking. For that reason, we will dedicate a complete section, Section 2.2, to the Conformance class, its characteristics and its properties.

- ***Refinement***
  In this case there is a-priori model, but this model does not conform the reality, i.e., there are some inconsistencies between the model and the log. This could happen because the idea of process the people responsible of building the model have in their heads does not correspond with what is really going on, or simply because the reality has changed and the model has become out-of-date. The aim of Process Mining Refinement techniques is to correct these deviations, refining the model to reflect as accurately as possible the reality.

  Notice that, the whole Chapter 7 of the work presented in this thesis is focused on Process Mining Refinement, proposing some algorithms to refine disconformant models, using the results of the conformance analysis techniques proposed in the Chapter 4.

- ***Extension***
  The last class of Process Mining is called Extension. In this case there is also an a-priori model, and the aim of the techniques is to *extend* this model with extra data or another perspective. Note that the goal is not to check conformance (as in Process Mining Conformance) but to enrich the model with the data in the event log.

  Examples of Extension techniques are the *Decision Miner* [32] (that analyzes how data attributes influence the choices made in the process) or the *Performance Analysis with Petri Nets plug-in* (that provides different Key Performance Indicators related to the execution of the process).

2.2 Process Conformance

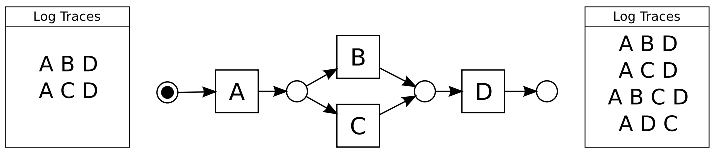
Given a model representing a process and a log reflecting the execution of this process, we are interested in analyzing quality of the model to describe the process (reflected in the log), i.e., is the model complying with the specified behavior, and is it done in a suitable way? Although the conformance between one log and one model could be considered the common scenario in conformance analysis, another scenario is also possible: the use of conformance analysis to compare between models, in order to determine which one represents better the behavior of the log, e.g., to compare the results of different Process Mining discovery approaches.

The final goal of the models is to describe *completely* the process they are modeling. But they also aim to describe it *precisely*, and in a *structurally* suitable way. In other words, the modeling of processes is a complex procedure involving several aspects. Conformance analysis must take into account all these aspects, or *dimensions* in order to obtain a complete and global vision of the model quality [29, 30].

There are four dimensions: *fitness*, *precision*, *generalization* and *structural*. In the following sections we present these dimensions in detail, providing examples and reviewing the previous work done to measure each dimension. We will especially focus on precision dimension, the final goal of the approaches presented in this thesis.

2.2.1 Fitness Dimension

Most of us, when we think about the quality of a model with respect to a log, we think of how much of the behavior represented in the log can be *captured* (*reproduced* or *replayed*) by the model, i.e., whether the reality (reflected in the log) fits the model. This aspect of the conformance process is called *Fitness Dimension*, and is perfectly illustrated in the example of the Fig. 2.5. This example shows the flow of a process modeled using Petri nets (cf. Sec. 3.2.1). When we compare this model with the first log (a), we observe that model is able to reproduce all the traces in the log, i.e., the fitness between the model and the log is perfect. However, when the same model is checked with the log (c), only 2 of the traces can be reproduced in the model, i.e., the model does not capture all the behavior reflected in the log.



(a) Log 1 (b) Model (c) Log 2

Figure 2.5: *Example of Fitness Dimension. The fitness between the log (a) and the model (b) is better than the fitness between (c) and the same model.*

Fitness is not the only dimension of the conformance process (there are three more), but it is the dimension most direct to see. And therefore, it is addresses by a huge number of approaches. Most of these approaches are based on the replay of the log on the model. This is the case of *ParsingMeasure* (PM) presented in [52]. PM is defined as the number of correct parsed traces divided by the number of traces in the event log. Similar to PM is the idea of *completeness*, presented in [18, 17], where the metric is defined as the percentage of traces in the log that are compliant with the model (a Workflow Schema in this case).

However, ParsingMeasure metric is a very naive approach that does not take into account the severity of the non reproducible traces, i.e., is not the same a trace that gets stuck in the last task during its replay than a trace that gets stuck in each event. Thus, the same authors present a more advanced metric called *ContinuesParsingMeasure* (CPM) [52]. This metric replays the log in a non-blocking way, and returns a measure based on the the number of successfully parsed events, the missing activities and the hanging activities during the replay. A similar idea is used in the fitness measure presented in [12]. This metric estimates the fitness between a Petri net and a log, calculating the number of times a transition that was supposed to be fired was actually enabled.

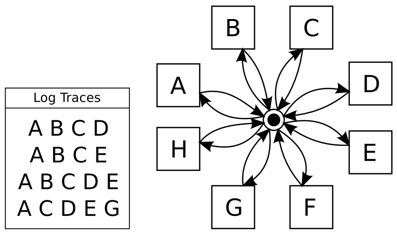
A more refined technique is the one used in *fitness* (f) [33] (and similar to $\mathrm{PF}_{\mathrm{complete}}$ [11]). This approach estimates the fitness between a Petri net and a log, based on the *token game*. The log is replayed in a non-blocking way, i.e., each time a task is not enabled, the missing tokens are created in order to enable it. At the end, the missing (i.e., artificially created) tokens are compared with the total number of tockens consumed during the replay, and the remaining tokens are compared with the produced ones, in order to estimate the degree of fitness.

As in fitness (f) seen above, the metrics proposed in [34, 35] computes the fitness between a Petri Net and a log. However, in this case, they use the Hidden Markov Models theory to estimate the fitness.

A different approach to address the fitness dimension is the one proposed in [9] (but it can also be seen as precision and generalization dimension too). This approach estimates the quality of a model comparing the *distance* between the traces in the log and the ones produced by the model. The authors propose several metrics to measure this distances such as the *Simple String Distance Metric*, the *Nonlinear String Distance Metric* or the *Event type Weighting Metric.*

2.2.2 Precision Dimension

To qualify a model as *good*, fitness is not the only requirement. Apart from capturing all the traces, a good model must avoid to represent more behavior than the one reflected in the log, being as close as possible to the reality. The *Precision Dimension* refers to the behavior allowed by the model, but never observed in the process executions in the log. A model including much more behavior than the log would become less informative, losing the power to describe accurately the process we want to model. The extreme case is the *flower model*, shown in Fig. 2.6(b). This model is able to capture all the traces in the log 2.6(a). However, it does not provide any useful information about the flow of the process reflected in the log. Actually, this same model is able to capture any possible log involving the tasks A,B,C,D,E,F,G and H.



(a) Log (b) Model

Figure 2.6: *Example of Precision Dimension. The model (b) is able to reproduce all the traces in the log (a), but it describes the process imprecisely, i.e., it allows much more behavior than the one reflected in the log.*

There are different approaches to quantify the Precision Dimension. In [33] the authors present two metrics to measure the precision of a Petri net with respect to the log. The first one, called *Simple behavioral appropriateness* (a_B), is based on the idea that an increase of enabled tasks during the log replay denotes an increase of alternatives and parallelism, and therefore an increase of the potential behavior reducing the precision. The second metric, called *Advanced behavioral appropriateness* (a'_B) is based on comparing model and log relations. First, *sometimes precedes* and *follows* relations from log and model perspective are computed. Then, the relations of the log and the model are compared with each other to find discrepancies, indicating possible variability in the model behavior not contemplated in the log. The main problem of this metric is the cost of the model state space exploration needed to derive the model relations.

A similar scenario is considered in [34, 35], where the precision between a Petri net and a log is checked. However, in this case, the authors use *Hidden Markov Models* in order to quantify the conformance, and a precision metric is provided according to it.

Other approach for measuring precision is *Behavioral Precision (B_P)* presented in [12]. This metric is used to compare two Petri nets (a reference and a compared one) on the basis of a log. Behavioral Precision checks how much behavior is allowed by the compared model that is not by the reference model. This is done comparing the number of enabled tasks in each model in every moment of the log. Additionally, this metric takes into account how often a trace occurs in the log, giving a higher weight to the most common behavior, in contrast to other infrequent and less important situations. The idea of Behavioral Precision is used in the Genetic Miner

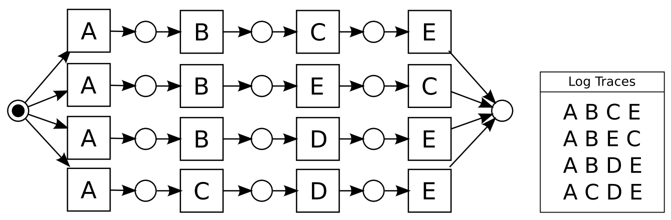
[14], to compute the precision of a mined model with respect to the reference model.

In *Causal Footprint* [49] the authors present an approach to check precision (but also generalization) without a log, based on the idea of detecting common erroneous patterns using *casual footprints*, i.e., a representation of a set of conditions on the order of activities that holds for every case of a process. This idea is applied to Petri nets and EPCs, and could be also applied to other languages as BPEL or UML diagrams.

A different idea is used for the approach presented in [18] and [17], called *soundness*, and used for checking the precision of a Workflow Schema with respect to a log. Soudness measures the percentage of traces that can be generated by the model that are in the log. Finally, in [6], the authors propose an independent-model approach to compare an event log and a model, based on the *minimal description length* principle.

2.2.3 Generalization Dimension

The *Generalization Dimension*, as the Precision Dimension seen above, refers to how much more behavior is captured in the model than the one observed in the log. However, in contrast to Precision, Generalization addresses overly precise models, measuring the overfitting. In Fig. 2.7 we can see an example of low value of generalization. The model of that example represents each trace of the log in a different path, avoiding any kind of generalization, and overfitting the given log.



(a) Model (b) Log

Figure 2.7: *Example of Generalization Dimension.*

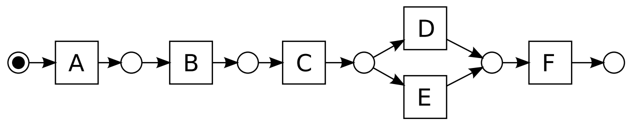
Measuring Generalization could be a complicated task, especially when noisy data and unstructured processes are involved. The abstraction of the less frequent behaviors during the construction of the model can lead to a loss of fitness and precision. Thus, Generalization must be seen as a single dimension by itself, and it must be balanced with the other dimensions to obtain a complete vision of the model quality.

Unlike Fitness and Precision Dimensions, in the literature there are few approaches addressed to Generalization Dimension. And even less if we consider this dimension alone, because some of the approaches perform the generalization and precision analysis together (this the case of *Causal Footprint* [49] or the process validation of [9], both explained above).

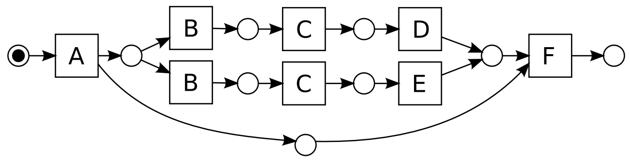
An example of metric addressing exclusively generalization is *Behavioral Recall (B_R)* [12]. This metric is used to compare two models on the basis of a log. Behavioral Recall checks how much behavior is allowed by the reference model that is not by the other model. This is done comparing the number of enabled tasks in each model in every moment of the log. Additionally, this metric takes into account how often a trace occurs in the log, giving a higher weight to the most common behavior, in contrast to other infrequent and less important situations. Behavioral Recall is the symmetric measure of the metric Behavioral Precision (B_P) seen adobe, and it is also used in the Genetic Miner [14], to compute the generalization of a mined model with respect to the reference model.

2.2.4 Structural Dimension

Finally, the last dimension considered in the conformance analysis is the *Structural Dimension*. Given a model describing a process, we want this model to be as understandable, meaningful and compact as possible. Understandable, meaningful and compact are subjective terms, and their meaning depend on the final purpose of the model and the modeling language used. For instance, the two models in Fig. 2.8 could be a good example to illustrate the Structural Dimension for the Petri net case. Both models express the same behavior. However, the model of (b) contains some elements that *inflate* its structure, e.g., it contains a redundant place that does not modify the behavior of the net, or some duplicate tasks representing the same step in the process and preventing to identify this step as unique (and not as two different activities).



(a) Model 1



(b) Model 2

Figure 2.8: *Example of Structural Dimension. Both models describe the same behavior. However, the structure of (b) is worst, i.e., it contains some elements that "inflate" the structure unnecessarily.*

There are some examples of metrics and approaches addressed to measure the Structural Dimension. In [33] the authors presents two metrics for the Petri net case: *Simple Structural Appropriateness* (a_S) and *Advanced Structural Appropriateness* (a'_S). The a_S metric uses the number of different transition labels (two duplicate tasks share the same label) and compare it with the model size to estimate the structural accuracy of the model. However, that metric can only be used as a comparative means for models exhibiting equivalent behavior because it is only based in the model size. To check the structural accuracy independently of the model behavior, the same authors propose a'_S, a more advanced metric that uses, as estimator, the verification of structural patterns or guidelines, such as the set of redundant invisible tasks or the set of alternative duplicate tasks.

In [14], the authors present two metrics, called *Structural Precision*(S_P) and *Structural Recall*(S_R), in order to check the structural conformance between a reference model and a mined model. Both metrics are based on quantifying how many connections have the models in common: S_P counts how many causality relations has the mined model that are not in the reference model, and S_R works the other way around. A similar idea is applied in *precision* and *recall* [28], in this case for comparing two Workflow graphs. Precision indicates the ratio of correctly identified edges over the total number of generated edges, and recall is the ratio of correctly identified edges over the total number of edges in the reference workflow graph. Finally, the metrics *Duplicates Precision* (D_P) and *Duplicates Recall* (D_R) [11] are similar to S_P and S_R, but quantifying how many duplicate tasks have in common the mined and the reference model.

3 Logs and Models

In this section we introduce the main elements involved in any Process Mining technique: the log (Sec. 3.1) and the model (Sec. 3.2). In Section 3.3 we introduce the mapping between these two elements in order to perform a conformance analysis. Finally, in Sec. 3.4 we present the Transition Systems, a component used later in the algorithms presented in this thesis.

3.1 Event Log

The *event log* [45] (also known as *workflow log* or simply *log*) is the main element of the Process Mining techniques. These logs contain information about the events and processes occurred in the system, and can be used to mine a model, to extend a existent one, or to measure its quality. Nowadays, any software system used for supporting processes (such as ERP[1] or WFM[2] systems) record this information in some form. For Process Mining purposes, the events recorded in the logs must satisfy that:

- each event refers to a *task* (i.e., a well-defined step in the process)
- each event refers to a *case* (i.e., a process instance)
- the events are totally ordered

Apart from that, the log might also include some other information, e.g., the timestamp of each event, the performer of each action, or some additional data. In addition, the log might contain the *type* of each event, e.g., *start event* (the beginning of an action), *complete event* (the end of an action), *scheduled event* (an action scheduled to be performed), etc. Table 3.1 shows an example of event log with some of this information.

The format used by each system to store the logs is usually different. The absence of a common format makes difficult to share and exchange logs among tools, increasing the cost of Process Mining analysis. For that reason, in 2003, a new format was proposed to become a standard for event logs: the *Mining XML* (or MXML) [45]. MXML is based on XML [1], and can be used to store any of the information related to the events and processes [3]. MXML is

[1]Enterprise Resource Management systems
[2]Workflow Management systems
[3]For more details, the XML Schema of MXML format can be seen in [2].

Case	Task	Type	Performer	Timestamp
...	...	...	...	...
1	Check Availability	complete	George	2010/09/04 17:13
2	Check Availability	complete	Alex	2010/09/04 17:14
3	Check Availability	complete	Dolores	2010/09/04 17:17
3	Ship Order	complete	Dolores	2010/09/04 17:21
1	Purchase Material	complete	George	2010/09/04 17:22
2	Ship Order	complete	Alex	2010/09/04 17:24
4	Check Availability	complete	Emily	2010/09/04 17:33
2	Send Bill	complete	Alex	2010/09/04 17:36
2	Check Payment	complete	Alex	2010/09/04 17:38
4	Ship Order	complete	Emily	2010/09/04 17:44
1	Register Purchase	complete	George	2010/09/04 17:50
3	Check Payment	complete	Dolores	2010/09/04 17:52
...	...	...	...	...

Table 3.1: *Example of Event log.*

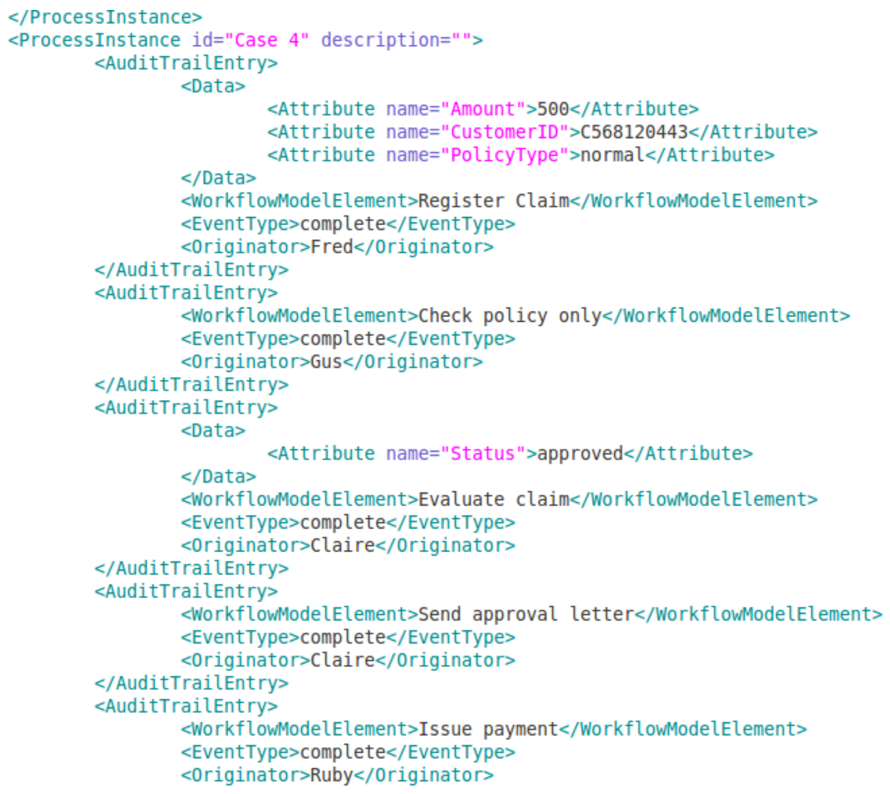
the format supported by most of the Process Mining analysis tools, such as *ProM* [4], the most important framework in the Process Mining area. It also exists a tool, called *ProM Import* [5], able to convert from almost any log format into MXML. Fig. 3.1 shows an example of event log fragment in MXML format.

For the purpose of the work presented in this thesis, not all the information contained in the log is needed. Only the one concerning about the control flow perspective, i.e., the task and the case of each event, and the order between events. Therefore, we can simplify the representation of the logs as it is shown in Fig. 3.2. In this simplification, each task is represented with a capital letter (e.g., *A* for *Check Availability*). Moreover, each line of the log (called *trace*) represents sequence of events grouped by case exhibiting the same event sequence. The number at the beginning of each trace denotes the number of instances combined together in that trace (the absence of number represents 1). It is important to remark that this simplification and this aggregation of the log can be done because only the control flow perspective is considered here. In case of considering other perspectives, the additional data stored in the log would play a more important role (such as *performer* in social networks perspective, or *timestamp* in performance analysis perspective).

A formal definition of trace and event log for that simplification can be found in Def. 3.1.

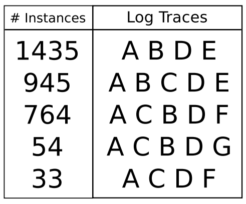
Definition 3.1 (Trace, Event Log) *Let T be the set of tasks, and let $\mathcal{P}(S)$ denote the powerset over S, i.e., the set of possible subsets of elements of S. A* trace *σ is defined as $\sigma \in T^*$. And an* event log *is a set of traces, i.e., $\mathsf{EL} \in \mathcal{P}(T^*)$.*

Finally, notice that the event log might contain sensitive information concerning the users of the system, such as its name or the time spent performing an action. The management of this information is usually regulated by law, and its use for analyzing aspects such as the performance of the employees could become a crime. Therefore, techniques to anonymize the event log could be a necessary action to take before starting any Process Mining analysis.



```
</ProcessInstance>
<ProcessInstance id="Case 4" description="">
        <AuditTrailEntry>
                <Data>
                        <Attribute name="Amount">500</Attribute>
                        <Attribute name="CustomerID">C568120443</Attribute>
                        <Attribute name="PolicyType">normal</Attribute>
                </Data>
                <WorkflowModelElement>Register Claim</WorkflowModelElement>
                <EventType>complete</EventType>
                <Originator>Fred</Originator>
        </AuditTrailEntry>
        <AuditTrailEntry>
                <WorkflowModelElement>Check policy only</WorkflowModelElement>
                <EventType>complete</EventType>
                <Originator>Gus</Originator>
        </AuditTrailEntry>
        <AuditTrailEntry>
                <Data>
                        <Attribute name="Status">approved</Attribute>
                </Data>
                <WorkflowModelElement>Evaluate claim</WorkflowModelElement>
                <EventType>complete</EventType>
                <Originator>Claire</Originator>
        </AuditTrailEntry>
        <AuditTrailEntry>
                <WorkflowModelElement>Send approval letter</WorkflowModelElement>
                <EventType>complete</EventType>
                <Originator>Claire</Originator>
        </AuditTrailEntry>
        <AuditTrailEntry>
                <WorkflowModelElement>Issue payment</WorkflowModelElement>
                <EventType>complete</EventType>
                <Originator>Ruby</Originator>
```

Figure 3.1: *Fragment of Event Log in MXML Format. The fragment corresponds to* exampleLog *available in [4]. In MXML format, each event is represented by the tag* AuditTrailEntry*, and the task associated with that event is identified by the tag* WorkflowModelElement*.*



# Instances	Log Traces
1435	A B D E
945	A B C D E
764	A C B D F
54	A C B D G
33	A C D F

Figure 3.2: *Example of Event Log Simplified.*

3.2 Model

The *model* (also called *process model*) is the other important element of all Process Mining techniques. A model is an abstracted representation that describes how a process really works. It can also describe the people or the organizations involved in the performance of the process. The model is used to analyze the process, gaining insight in it, and making easier to take decisions concerning about the process.

A process model could be the result of a Process Mining technique (*Discovery* in this case), or the result of a manual design, i.e., a group of designers with deep knowledge of the modeling language and the process itself construct a detailed model that describes accurately the reality. On the other hand, it could exist an *a-priori* model (designed or discovered), and the aim of the

Process Mining is to extend it (*Extension*), to measure its quality (*Conformance*) or to correct it (*Refinement*). These last two scenarios are the ones considered in this thesis.

There are a wide variety of modeling languages, and the decision of choosing one or another depends on many factors. One of these factors is the perspective, i.e., there are different modeling languages for the different perspectives. For example, the use of *Petri nets* or *Event-driven Process Chain (EPC)* for control flow perspective, or *Social Networks Graphs* for organizational perspective. Some of the other factors that influence the choice of a language are the formalism required, the visual representation of the model, or the existing tools for design and analyze the chosen modeling language.

In the following section we introduce and describe the modeling language chosen for the approach proposed in this thesis: Petri nets.

3.2.1 Petri Nets

The use of Petri nets [26, 27] for process flow modeling is common in Process Mining, and the list of miners designed to discover Petri nets from event logs is long [48]. Petri nets provides a formal semantic, a solid mathematical base, and a graphical notation, making them appropriate for modeling processes. Moreover, ProM [4] framework provides a wide range of tools for converting other modeling languages into Petri Nets. Note that, although the approach presented in this paper is based on Petri Nets, the idea can also be applied to other modeling language with executable semantics.

In the remaining of this section we present a formal definition for Petri nets, an some of the terminology used in Petri net theory.

Definition 3.2 (Petri Net [26]) *A Petri Net (*PN*) is a tuple* (P, T, W, m_0) *where* P *and* T *represent finite sets of places and transitions, respectively, with* $P \cap T = \emptyset$*. In addition, the relation* $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ *defines the weighted flow relation. A marking is defined as a mapping* $P \to \mathbb{N}$*.* m_0 *is the initial marking, i.e. defines the initial state of the system.*

A transition $t \in T$ is *enabled* in a marking m iff $\forall p \in P : m(p) \geq W(p, t)$. An enabled transition can be *fired* resulting in a new marking m' such that $\forall p : m'(p) = m(p) - W(p, t) + W(t, p)$. A marking m' is *reachable* from m if there is a sequence of firings $\sigma = t_1 t_2 \dots t_n$ that transforms m into m', denoted by $m[\sigma\rangle m'$. A sequence of transitions $\sigma = t_1 t_2 \dots t_n$ is a *feasible sequence* if $m_0[\sigma\rangle m$, for some marking m. When a sequence σ is a feasible sequence of PN, we say that the Petri net PN is able to *reproduce* (or *replay*) the trace σ. The sequences $t_1 t_2 t_5$ and $t_1 t_2 t_7$ are examples of traces reproducible and non reproducible for the Petri net 3.3 respectively. We define the *language* of a Petri net $\mathrm{L}(\mathsf{PN})$ as the set of all feasible sequences. The set of reachable markings from m_0 is denoted by $[m_0\rangle$, and form a graph called *reachability graph.* A PN is said to be *k-bounded* or simply *bounded* if $\forall p : m'(p)$ does not exceed a number k for any reachable marking m' from m_0. If no such number exists, it is said to be *unbounded.* Fig. 3.3 shows an example of Petri nets graphical representation, indicating each of its elements.

3.3 Mapping between Model and Log

The first step in order to perform any action involving both model and log is to establish a relation between both objects, i.e., the events of the event log and the transitions of the Petri net must be mapped. The association relation presented in this section is similar to the one proposed in [31]. The mapping should be performed by a domain expert, with knowledge about the analyzed process, the log and the model, in order to associate correctly events and

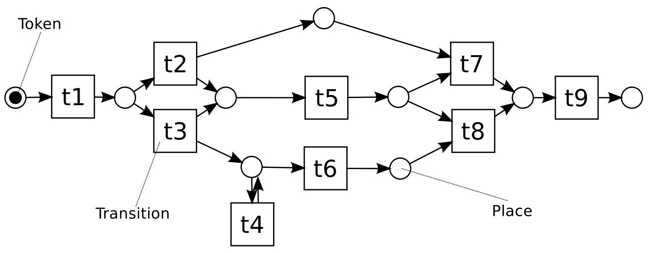


Figure 3.3: *Elements of a Petri net: Transitions, Places and Tokens (the number of tokens indicates the number associated with that place in the current marking).*

transitions. Depending on relation between elements we can distinguish the following types of associations:

- ***Ordinary tasks***
  The most simple and common relation between events and transitions: the 1-to-1 mapping, i.e., each event in the log is associated with only one transition in the Petri net. In this case, the transition is labeled with the name of the event.

- ***Duplicate tasks***
  In this case, two or more transitions in the model are associated with the same event of the log. All transitions belonging to a duplicate tasks relations are labeled with the same event name. However, from a strictly model point of view they remain being differentiable elements of the model with its own position in the Petri net. The reverse case (i.e., one transition associated to multiple events) may happen if the execution of the process is recorded at a more fine-grained level, e.g., start and end annotations at the beginning and ending of the task execution. However, as it was mention before, we abstract from this fine-grained information considering only *complete* actions, maintaining the approach as general as possible.

- ***Invisible tasks***
  A transition in the model is associated with no event in the log, i.e., actions of the process with no track in the log. The reasons for this lack of relation may be different, e.g., unrecordable steps in the process (phone calls, meetings, ...), introduced artificially in the model for routing purposes, relaxation of the models (we only focus in the log of a part of the model, non concerning about rest of the model), or as a result of some discovery algorithm (e.g., Genetic Miner [42]). The transitions belonging to this relation are not labeled, and they are represented as black filled rectangles.

- ***Non modeled tasks***
  This relation refers to the events of the log not related with any transition in the model. For the purposes of this work we assume that the log can be preprocessed, removing this type of events before starting the analysis.

The techniques presented in this paper cover all the scenarios above. Note that, for the sake of clarity, in this thesis we will refer to task, event or transition indistinctly, whenever no mistake is possible.

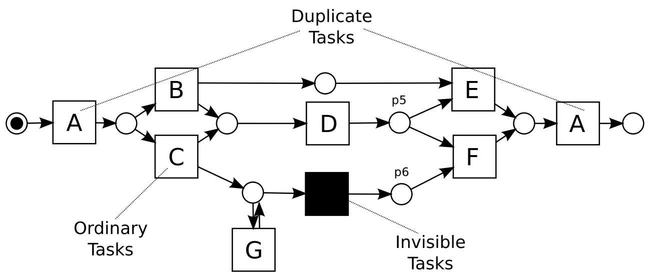


Figure 3.4: *Mapping Relations between Petri nets and Event Logs: Ordinary tasks, Duplicate Tasks and Invisible Tasks.*

3.4 Transitions Systems

The last formal model presented in this chapter are *transition systems.* Transitions Systems are widely used in Process Mining, because their flexibility allows to represent other models and they properties, such as the sequence of events reflected in a log, or the possible behavior of a Petri net. Actually, sometimes are the transition systems themselves the ones that are used as input of some Process Mining algorithm. An example of that is *Genet* [7], a tool that allows the derivation of a general Petri net from a transition system representation of a system. In other cases the transitions systems are the results (or outputs) of the algorithms, as in [44], where the authors propose methods to derive a transition system from an event log. However, in the work presented in this thesis the transition systems will not be used as inputs or outputs, but as intermediate elements to develop the algorithms proposed.

Definition 3.3 (Transition system) *A* transition system *(or* TS*) is a tuple* (S, T, A, s_{in})*, where* S *is a set of* states*,* T *is an alphabet of* actions*,* $A \subseteq S \times T \times S$ *is a set of* (labeled) transitions*, and* $s_{in} \in S$ *is the* initial state*.*

We will use $s \overset{e}{\to} s'$ as a shortcut for $(s, e, s') \in A$, and the transitive closure of this relation will be denoted by $\overset{*}{\to}$. The *language* of a transition system TS, $L(\mathsf{TS})$, is the set of traces feasible from the initial state.

4 Process Conformance Approach

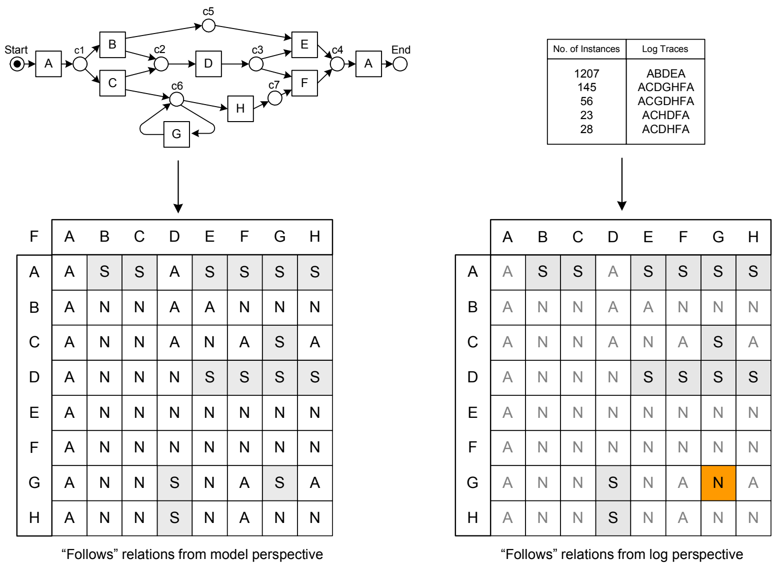
In this chapter we present our approach for checking precision between a Petri net and an Event log. In Sec. 4.1 we explain the general idea of the approach, its motivations, the *route map* of the procedure, and we compare it with similar approaches. In Sec. 4.2 and Sec. 4.3 the procedure to check the precision is explained in detail. In Sec. 4.4 and Sec. 4.5 we present the two elements resulting of the precision analysis: the ETC Precision metric and the set of Minimal Disconformant Traces. Finally, in Sec. 4.6 some special cases, variants and extensions are considered[1].

4.1 General Idea and Approach Route Map

The aim of this chapter is to propose an approach to measure precision between Event Logs and Petri nets. The most similar work with this same purpose is [33], by A. Rozinat and W.M.P. van der Aalst. In that work, the authors present several metrics and methods to measure the different dimensions of the conformance between logs and Petri nets. In particular, they present a metric for quantifying precision called a'_B. For computing this metric, first we need to build the *Precedes Relation* and the *Follows Relation* between tasks, for both model and log perspective. These relations can only take the values of *Always*, *Sometimes* and *Never*. Then, we compare the relations of the model and the log, counting the Sometimes relations of the model that does not appear in the log. These discrepancies may point out precision problems (some variability of behavior in the model not reflected in the log) and are used to compute the metric a'_B (cf. Fig. 4.1). The problem of this approach is that to build the relations from a model perspective, we need to analyze the possible executions sequences of the model, i.e., we need to explore the state space of the model. Because the state space of a model can grow exponentially, state-based techniques may be problematic for real case scenarios.

The purpose of this chapter is to present a new technique for measuring precision in Petri nets that does not require the exhaustive exploration of the model state space. This approach can be summarized as follows: given a model and a log, the behavior of the model restricted to the log is computed (dark gray part of Fig.4.2). The border between the log's and model's behavior defines crucial points where the model deviates from the log. We call these situations *escaping edges*. By quantifying these edges and their frequency, we aim at providing an accurate measurement of the precision dimension. Moreover, the escaping edges denote inconsistencies

[1]Most of the ideas of this chapter appear also in the paper [25].


Start
A
c1
B
C
c2
c5
D
c3
c6
G
H
c7
E
F
c4
A
End
"Follows" relations from model perspective
"Follows" relations from log perspective

No. of Instances	Log Traces
1207	ABDEA
145	ACDGHFA
56	ACGDHFA
23	ACHDFA
28	ACDHFA

F	A	B	C	D	E	F	G	H
A	A	S	S	A	S	S	S	S
B	A	N	N	A	A	N	N	N
C	A	N	N	A	N	A	S	A
D	A	N	N	N	S	S	S	S
E	A	N	N	N	N	N	N	N
F	A	N	N	N	N	N	N	N
G	A	N	N	S	N	A	S	A
H	A	N	N	S	N	A	N	N

"Follows" relations from model perspective

	A	B	C	D	E	F	G	H
A	A	S	S	A	S	S	S	S
B	A	N	N	A	A	N	N	N
C	A	N	N	A	N	A	S	A
D	A	N	N	N	S	S	S	S
E	A	N	N	N	N	N	N	N
F	A	N	N	N	N	N	N	N
G	A	N	N	S	N	A	N	A
H	A	N	N	S	N	A	N	N

"Follows" relations from log perspective

Figure 4.1: *"Follows" relations derived from model and log perspectives in order to calculate the a'_B metric. The only discrepancy in this case correspond with the task G. The "Precedes" relations would be computed similarly. This figure was obtained from [33].*

that might be treated in the process refinement phase.

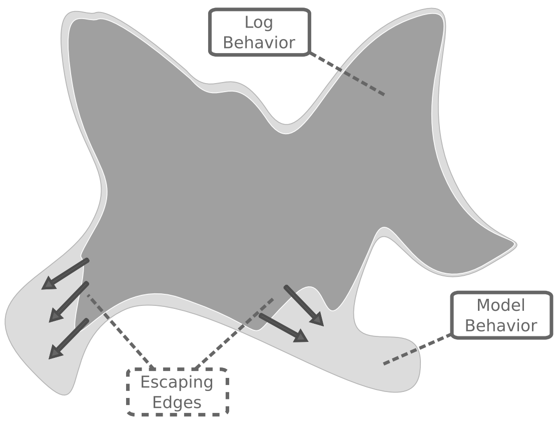


Figure 4.2: *Diagram of Log-based traversal of Model Behavior. This traversal does not explore all the model behaviors (light gray), only the part associated with the log behavior (dark gray). The Escaping Edges denote the points where the model behavior deviates from the log behavior.*

Some of the characteristics of this new approach are the following:

A fresh look at precision
We aim at providing a precision metric that estimates the effort needed to obtain an accurate model, focusing on the discrepancies detected. This contrasts with the existing approaches for precision that only provide discrepancies in the event relations [33]. The key concept of the approach is that of *Escaping Edges*, i.e., the situations were the model allows more behavior than the log, thus exhibiting less precision. We base our measure on the relation between the

number and frequency of escaping edges with respect to model's behavior restricted to the log. Hence, the more the model deviates from the log, the less precise is its precision value. It is important to stress the fact that a model can have few escaping edges (thus exhibiting good precision) but with underlying behavior substantially bigger than the log: it is not the aim of this work to compare sizes between the model and the log, but providing an estimation of the efforts required to improve the model to precisely describe the log.

Efficiency
In contrast to other approaches that require a complete exploration of both the model and the log behavior [33], our approach performs a log-based traversal of the model behavior, i.e., the model exploration is restricted to the observed log's behavior. As a consequence, the computational requirements are bounded to the log size, thus being independent of the whole model underlying behavior. This might be crucial for models obtained from a mining algorithm or real case scenarios that may have an underlying behavior of intractable size.

Fine Localization
Our approach works directly with the behavior of the model, and not with the model itself. As results, we get a deeper view of the precision, being able to locate the exact moment and place of the precision issues. On this regard, the methods presented in this chapter output also the exact points of discrepancy, i.e., the traces where the model starts to deviate from the log. This accurate localization may be a good starting point for correcting and refining the model.

The procedure of checking the precision of a system using this approach can be divided in several stages. Figure 4.3 shows the route map of the procedure. In the first stage, some state information is obtained from both the model and the log, in order to obtain a common domain between log and model behaviors (Section 4.2). Once this is done, we use this information to perform the log-based traversal of the model behavior(Section 4.3). Finally, we use the information recollected during the traversal to generate and output two results: the first one, called *ETC Precision* (etc_P), is a metric that quantifies the level of precision of the system (Section 4.4); the second one, called *Minimal Disconformant Traces (*MDT), is an element designed to point out with exactitude the origins of the precision issues (Section 4.5).

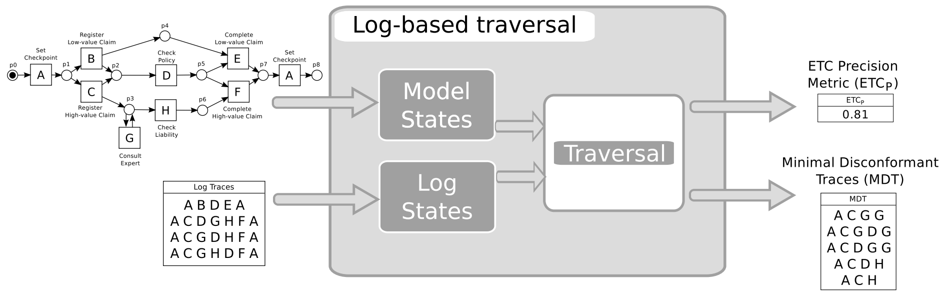
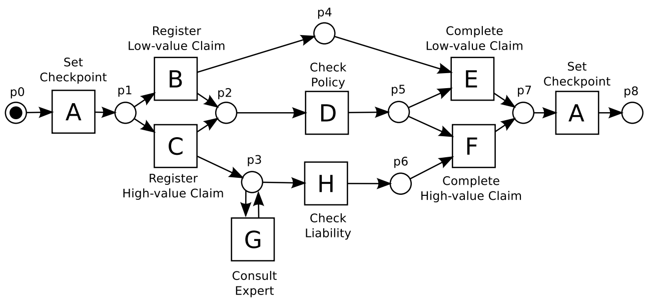


Figure 4.3: *Route Map of the Precision Approach.*

During the precision checking presented in this section, we assume that every trace in the log is possible in the model, i.e., the model overapproximates the log, being 1 the fitness value of the system. This is the scenario shown in Fig.4.2. The reverse case, i.e., a model more specific than the log, indicates not a precision problem but a fitness problem. In Section 4.6.3, we discuss how to adapt the technique to deal with non-fitting models. In addition, for the sake of clarity and understanding, during the first part of the section we will assume the most simple mapping between transitions of the Petri net and events of the Event log., i.e., without invisible or duplicate tasks. In Section 4.6.1 we extend the general approach to include these special cases, and we analyze the consequences of this extension.

The Running Example
During this chapter, in order to illustrate each one of the steps performed by the approach, we will use the example shown in Fig. 4.4 as a running example. This example is based on the ones used in [33]. The figure represents a banking system scenario. In particular, it reflects the typical process of liability insurance claim in a bank. The example is composed by the log and the model of the process. The log, shown in Fig. 4.4(b), reflects four executions of the liability process, each one represented as a trace of the log. Note that, for the sake of clarity, each task of the log is represented by a capital letter, e.g., *A* represents the *Set Checkpoint* task, and so on. On the other, Fig. 4.4(a) shows the model representing the control flow of the liability process (a Petri net in our case). Note that, although the log represents a plausible situation, the control flow shown in the model is not realistic, i.e., the *Consult Expert* task should be executed exactly once (we are assuming it make no sense to consult an expert more than one time). This loop reflects an extra behavior allowed by the model but not reflected in the reality represented by the log, i.e., a precision issue. We will use this issue to illustrate the precision checking performed by the approach presented in this thesis. The detection and localization of this inconsistence may seem obvious in this small scenario, but may be not for larger and realistic cases, including complex processes and a huge number of tasks.



(a) Model

Log Traces
A B D E A
A C D G H F A
A C G D H F A
A C G H D F A

(b) Log

Figure 4.4: *Running example to illustrate the conformance approach. The example contains a Petri net modeling the control-flow of a banking process, and a log with four executions of this process.*

4.2 Log States and Model States

In order to perform a log-based exploration of the model behavior, it is necessary to find a common comparable domain between log and model, i.e., a domain where the situations reflected in the model and the ones reflected in the log could be mapped. For performing such task state information must be derived from both objects: model and log.

In the log side, state information can be obtained using the method presented in [44]. Given a position in the log, this method proposed three parameters to define the current state: *time*, *horizon* and *representation*. The time parameter defines which group of tasks are used to define the current state, i.e., the *past* (the tasks before the current position), the *future* (the tasks after the current point), or *both* (past and future together). The Fig. 4.5 shows examples for the past and the future configuration. The second parameter is the horizon, i.e., the number of tasks considered to derive the state information. This parameter could be a number (the n tasks closer the the position are the ones that are considered), or could be ∞ (no horizon considered). Fig. 4.5 shows examples of two different horizons, and the tasks considered in each case. The last parameter, the *representation*, defines how the state information is represented: as a *sequence* (the order of the tasks used to define the state matters), as a *set* (the order does not matters), or as a *multiset* (it considers the number of times of each task, ignoring their order). For instance, if sequence option is selected, states defined by chains "AB" and "BA" are different states; however, choosing "set" configuration, both will refer to the same state (the one defined by the set containing A and B).

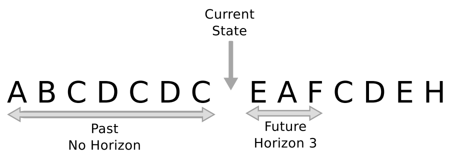


Figure 4.5: *Example of the parameters used to derive state information from a running example log. The example shows the tasks considered to define the state in two configurations: Past with no Horizon, and Future with Horizon 3.*

In our case, the use of the past, sequence and infinite horizon settings allows to derive a behavioral representation of the log (a transition system) with the same language. The states of that transition system (or automaton) will be the sates of the log.

Definition 4.1 (Prefix automaton, Log states) *Given an event log* EL*, let* $\mathsf{TS} = (S, T, A, s_0)$ *be the transition system derived by the method presented in [44] using the past, sequence and infinite settings. We call this transition system* prefix automaton*. The set* S *will be denoted as the* set of states of EL*. Given a trace* $\sigma_i = t_1 \dots t_{|\sigma_i|} \in \mathsf{EL}$*,* $s^i_j \in S$ *denotes the state in* TS *corresponding to the prefix* $t_1 \dots t_{j-1}$*, for* $0 \leq j \leq |\sigma_i| + 1$*.*

Following with the running example, Fig. 4.6 shows the prefix automaton derived from the log of Fig. 4.4(b). The states of the transition system compose the states of that log, and each state corresponds to prefixes of the traces in the log. For instance, the state 13 corresponds to the prefix ACG appearing in third and forth traces.

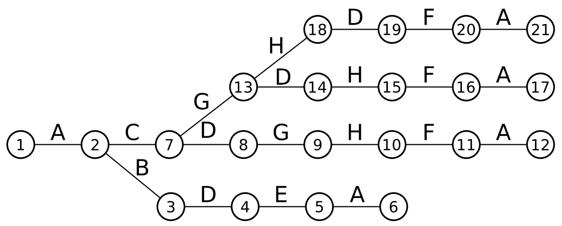
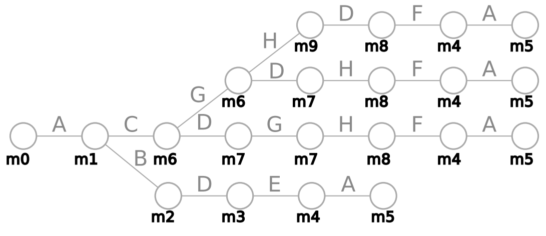


Figure 4.6: *Prefix automaton derived from the log of Fig. 4.4(b) using the past, no horizon and sequence configuration. The number inside correspond to the identifier of the state. The states composing this automaton correspond with the states of the log. For instance, the state 13 correspond to the state defined by the prefix "ACG" that appears two times in the log.*

In the model side, a Petri net in our case, state information can be obtained straightforwardly computing the set of possible markings of the net. However, due to the well-known *space-explosion problem*, Petri nets can exhibit a large or even infinite behavior, making this approach impractical for these instances. Instead, the approach presented in this paper only computes those reachable markings of the net for which there is at least one state in the log mapped. Let us define formally the mapping:

Definition 4.2 (Mapping between Log States and Petri net States) *Let* EL *be a log and* PN $= (P, T, W, m_0)$ *a Petri net, and consider the prefix automaton* TS $= (S, T, A, s_0)$ *from* EL*. A marking m is mapped to the state $s \in S$, denoted by $m \multimap s$, if there exists a trace σ such that $s_0 \xrightarrow{\sigma} s$ in* TS *and $m_0[\sigma\rangle m$.*

Following with the running example, Fig. 4.7 shows the mapping between the log states and the model states. For instance, the marking of the state with prefix AB is the one that contains one token in the place p2, one in p4 and zero in the others. Note that, the same marking can be associated with different log states, i.e., the marking does not define the states, the prefix does (in Section 4.6.2 we discuss the option of define the states as markings). Due to this mapping, the Petri net traversal can be controlled to reach only markings for which there is a corresponding mapped state in the log, bounding the exploration.



(a) Mapping

id	tokens
m0	p0
m1	p1
m2	p2 p4
m3	p4 p5
m4	p7
m5	p8
m6	p2 p3
m7	p3 p5
m8	p5 p6
m9	p2 p6

(b) Markings

Figure 4.7: *Mapping between Log States and Model States. The marking associated to each log state corresponds to the marking reached after replaying in the model the state prefix. The table indicates the tokens associates to each marking, e.g., "p2,p3" indicates one token in place p2, one in p3, and zero in the other places.*

4.3 Log-based Traversal of the Model's Behavior

Once the log states and the model states have been defined, and the mapping between them has been established, the log-based traversal of the model's behavior can be performed. This exploration makes possible to compare the behavior of the model and the log, detecting the discrepancies existing between them, and obtaining a conformance result for the pair.

In order to detect the discrepancies between model behavior and log behavior, first it is necessary to define the concepts of *allowed tasks* (behavior allowed by the model in a given moment) and *reflected tasks* (all the behavior that really happen, and therefore is reflected in the log). A formal definition of these two concepts is the following:

Definition 4.3 (Allowed Tasks and Reflected Tasks) *Let s be a state of the prefix automaton* TS $= (S, T, A, s_0)$ *from* EL*, and let* PN $= (P, T, W, m_0)$ *be a Petri net. We define $A_T(s) = \{t \in T \mid m \multimap s \wedge m[t\rangle m'\}$ and $R_T(s) = \{t \in T \mid s \xrightarrow{t} s')\}$ as the set of allowed and reflected tasks in s.*

Figure 4.8 reflects the table of Allowed Tasks and Reflected Tasks for the running example of Fig. 4.4. For instance, state 2 (with prefix A) has two Allowed tasks, B and C. Both correspond with the enabled transitions of the model for the marking m1 (only one token in p1) associated with that state. Moreover, both B and C tasks are reflected from that state in different traces: prefix AB appear in the first trace, and AC in the others. Therefore, both AB and AC are behaviors reflected in the log. Since we are assuming fitness value one of the model with respect to the log, clearly $R_T(s) \subseteq A_T(s)$ for every state s of TS, i.e. the model overapproximates the log.

State ID	State Prefix	Marking ID	Marking Tokens	Allowed Tasks (AT)	Reflected Tasks (RT)	Escaping Edges (EE)
1		m0	p0	A	A	
2	A	m1	p1	B C	B C	
3	AB	m2	p2 p4	D	D	
4	ABD	m3	p4 p5	E	E	
5	ABDE	m4	p7	A	A	
6	ABDEA	m5	p8			
7	AC	m6	p2 p3	D G H	D G	H
8	ACD	m7	p3 p5	G H	G	H
9	ACDG	m7	p3 p5	G H	H	G
10	ACDGH	m8	p5 p6	F	F	
11	ACDGHF	m4	p7	A	A	
12	ACDGHFA	m5	p8			
13	ACG	m6	p2 p3	D G H	D H	G
14	ACGD	m7	p3 p5	G H	H	G
15	ACGDH	m8	p5 p6	F	F	
16	ACGDHF	m4	p7	A	A	
17	ACGDHFA	m5	p8			
18	ACGH	m9	p2 p6	D	D	
19	ACGHD	m8	p5 p6	F	F	
20	ACGHDF	m4	p7	A	A	
21	ACGHDFA	m5	p8			

Figure 4.8: *Table with the Results of the Traversal for the Running Example. The table includes: the state, the prefix associated to the state, the marking associated with the state,and the set of Allowed Tasks, Reflected Tasks and Escaping Edges of the state.*

Once the concept of allowed tasks and reflected tasks has been defined, the discrepancies between behaviors (called *escaping edges*) can be defined too.

Definition 4.4 (Escaping Edges) *Let s be a state of the prefix automaton* $\mathsf{TS} = (S, T, A, s_0)$ *from* EL*, and* $\mathsf{PN} = (P, T, W, m_0)$ *a Petri net. The Escaping Edges (E_E) of s is defined as* $E_E(s) = A_T(s) \setminus R_T(s)$.

The table with the Escaping Edges for the running example of Fig. 4.4 can be seen in Fig. 4.8. An example of escaping edge of this example could be H for the state 7 (with the AC prefix associated). In this case, the model allows the execution of H in this state, whereas this is not reflected in the log (the prefix ACH does not appear in any trace). It is important to stress that the tasks reflected of a state s are the ones that appear in any of the traces that contain s. For instance, the state 2 of the running example (prefix A) has two allowed tasks in the model (B and C) but given that both are reflected in the log (in different traces), no escaping edge arise from that state.

After defining the discrepancies between behaviors, an algorithm to collect all them is presented in Algorithm 1. This algorithm explores all the states of the log, computing the allowed and reflected tasks of each of them. With this two sets calculated, the algorithm compute the escaping edges of the state and store them for future conformance analysis.

```
**Input**: EL: Event Log ; PN: Petri Net
foreach State s in EL do
    $R_T$ := outEdges ($s$) ;
    $\sigma$ := prefix ($s$,EL) ;
    $mark$ := fire ($\sigma$,PN) ;              // Fire $\sigma$ and get the reached marking
    $A_T$ := enable ($mark$,PN) ;           // Get the enable transitions of $mark$
    $E_E$ := $A_T \setminus R_T$ ;                   // Allowed tasks minus Reflected Tasks
    register ($s$,$E_E$) ;
end
```

Algorithm 1: Compute Escaping Edges Algorithm

4.4 Evaluating the Precision

As it has been seen before, the escaping edges are a good indicator for measuring the precision of the model behavior compared to the behavior reflected in the log. For that reason, we propose a metric to take into account these escaping edges and their frequency. This metric also allows us to compare between models to know which one captures better the behavior reflected of a log. Let us formalize the metric:

Metric 4.1 (ETC Precision) *Let* $\mathsf{EL} = \{\sigma_1, \ldots, \sigma_{|\mathsf{EL}|}\}$ *and* $\mathsf{PN} = (P, T, W, m_0)$ *be a log and a Petri net, respectively. For each trace* σ_i $(1 \leq i \leq |\mathsf{EL}|)$*, state* s^i_j $(1 \leq j \leq |\sigma_i| + 1)$ *denotes the* $j - th$ *state of* σ_i *(see Def. 4.1). The metric is defined as follows:*

$$etc_P(\mathsf{EL}, \mathsf{PN}) = 1 - \frac{\sum_{i=1}^{|\mathsf{EL}|} \sum_{j=1}^{|\sigma_i|+1} |E_E(s^i_j)|}{\sum_{i=1}^{|\mathsf{EL}|} \sum_{j=1}^{|\sigma_i|+1} |A_T(s^i_j)|}$$

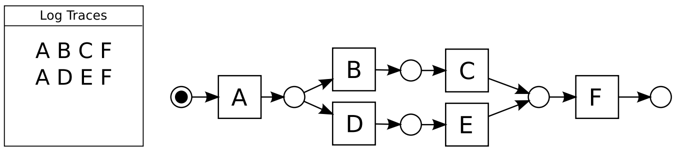
By dividing the set of escaping edges by the set of allowed tasks in the model, the metric evaluates the amount of overapproximation in each trace. The metric value for the running example in Fig. 4.4 is

$$1 - \frac{0 + 3 + 3 + 2}{6 + 12 + 13 + 12} = 0.81$$

where every i-th summand of the numerator/denominator is processing the penalizations for the escaping edges of trace σ_i, e.g., in trace σ_4 (ACGHDFA) of the log there are 2 escaping edges and 12 allowed tasks.

Note that, for all s^i_j, $|E_E(s^i_j)| \leq |A_T(s^i_j)|$, and therefore the values returned by etc$_P$ are distributed between 0 and 1. The more escaping edges, the lower value will be provided (even closer to 0 in the worst case). On the other hand, more precision between model and log, greater value of etc$_P$ is returned, being 1 if there are no inconsistencies.

Notice that, to achieve a value of 1 it is not necessary to have one and only one enabled task at each point of the trace (like in a_B [33]). This is because the metric does not depend on the idea of *more enabled tasks, more behavior*, but in the concept *behavior allowed vs reflected* itself. Therefore, the metric value can be 1 even if the whole behavior of the model is distributed in two or more different traces. This idea is illustrated perfectly in the example of Fig. 4.9. In this example there is a model with two *possible* paths: the upper path composed by A,B,C and F, and the lower path composed by A,D,E, and F. In addition, we consider the log in (a) that reflect the execution of each one of the possible paths (each path in a different trace). Given the notions of conformance and precision seen so far, it can be said that the model describes precisely the reality reflected in the log (there is no extra behavior, only the one really executed). And therefore, the etc_P value should be 1. And in fact, it is. This is because the decision of etc_P is made *globally*, i.e., the computation of the escaping edges is done globally after processing the whole set of traces, and not only considering the current trace.



(a) Log (b) Model

Figure 4.9: *Example of the global nature of etc_P. It evaluates the precision in a global way, i.e., can return 1 even if the whole behavior of the model is distributed in two or more different traces.*

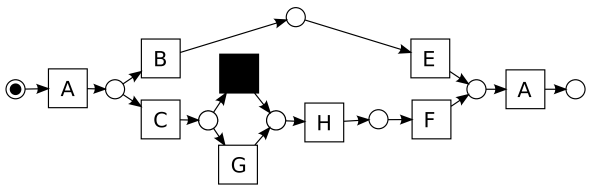
One of the strong points of this metric is the use of the trace frequency for computing the precision degree. There is no *sampling* process to the log: each trace in the log is used without mattering if there are more traces in the log equal to this one, or this is unique. The most common and appropriate traces that appear more times in the log are counted more times by the metric. Therefore, they will contribute with higher weight to the formula.

Another characteristic of the metric proposed in this section is the structural independence of the metric with respect to the analyzed model, i.e., it quantifies only the precision without being affected by the structural elements of the model. In other words, two models with different structure but with the same precision with respect to a log would have the same etc_P value. This is because the metric only focuses on the underlying behavior of the model, and not on the model itself. Figure 4.10 exemplifies this structural independence. It shows two models, (a) and (c), with the same behavior but different structure: (a) contains an invisible task, and (c) contains a duplicated task. However, the behaviors of the two models are the same, and therefore, the precision is the same with respect to the log (b). As a result, the etc_P value of both pairs model-log is the same (1 in this case).

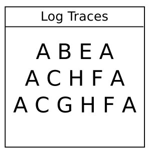
For being useful and complete, a metric must be able to *locate* those parts in the analyzed object that lack certain measured properties. This is a crucial requirement in conformance analysis, due to the fact that providing the discrepancy points we are making possible to identify the potential points of model enhancement. This is done in the approach of this thesis through the escaping edges, identified by their marking and the task used to *escape* from the reflected behavior in the log. However, given the importance of the localizability in conformance, we go a step further and in the next section we present a technique to collect the traces leading to these situations, called *Minimal Disconformant Traces*.

4.5 Locating the Precision Problems

As it has been pointed out previously, given a log and a model, we are not only interested in quantifying, but also in locating the precision discrepancies between them. One of the possible ways to identify this discrepancies are the *minimal disconformant traces*, i.e., minimal traces that lead to a situation where the model starts to deviate from the log. This kind of traces are



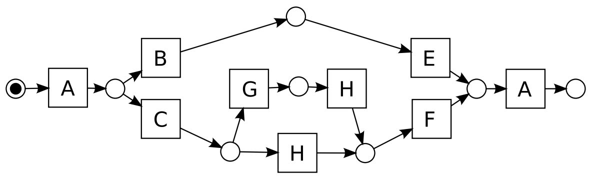
(a) Model 1



Log Traces

A B E A
A C H F A
A C G H F A

(b) Log



(c) Model 2

Figure 4.10: *Example of the Structural Independence of etc$_P$. Both models has different structure but the same behavior. Therefore, etc$_P$ should be (and it is) equal. In particular, the value is 1 with respect to the log in (b).*

specially useful to identify the origin of the possible problems in the model. A formal definition of these traces is presented in Def. 4.5.

Definition 4.5 (Minimal Disconformant Traces) *Let* EL *and* $\mathsf{PN} = (P, T, W, m_0)$ *be a log and a Petri net, respectively. Let* $\mathrm{Pref}(\mathsf{EL})$ *be the set of all prefixes for the traces in* EL*. We define the* Minimal Disconformant Traces *(*MDT*) as the set of traces* $\sigma = \sigma' t$ *such that* $m_0[\sigma\rangle M$*,* $\sigma' \in \mathrm{Pref}(\mathsf{EL})$ *and* $\sigma \notin \mathrm{Pref}(\mathsf{EL})$*.*

Given a log and a model, it is not necessary to compute the MDT from scratch. We can use the escaping edges computed in the precision analysis to build them. Algorithm 2 shows how to generate the MDT using the E_E: for each log state s with a escaping edge, the sequence to reach s is obtained (the *prefix* of the state s) and it is concatenated with the escaping edge.

```
Input: EL: Event Log
Output: $M$: Minimal Disconformant Traces
foreach State $s$ in EL do
    foreach Task $t$ in $E_E(s)$ do
        $\sigma$ := prefix ($s$,EL);                 // Get the prefix of the state $s$
        $\sigma := \sigma \cdot t$ ;                              // Concatenate
        addTrace ($\sigma$, $M$) ;                    // Register $\sigma$ as an MDT
    end
end
return $M$
```

Algorithm 2: Compute MDT Algorithm

Lemma 4.1 claims that the computed traces correspond to the Minimal Disconformant Traces defined above, ensuring the minimality criterion on the derived traces.

Lemma 4.1 *Algorithm 2 computes the Minimal Disconformant Traces.*

Proof: Let M the set of traces computed by the Algorithm 2 and MDT the set of traces that satisfy Definition 4.5. To prove $M = \mathsf{MDT}$, we will prove $M \subseteq \mathsf{MDT}$ and then $MDT \subseteq M$.

Let $\sigma = \sigma' t$ be any trace of M. By construction σ' is a prefix of the log. However, given the formation of R_T and E_E in Algorithm 1, σ is not a prefix of the log. Furthermore, σ' is a feasible sequence of the model because it is a prefix of the log, and all traces in the log are compliant with the model (since we assume fitness value one). In addition, by construction of A_T and E_E, σ is a feasible sequence by the model too. Therefore, $\sigma \in \mathsf{MDT}$.

Now, let $\sigma = \sigma' t$ be any trace of MDT. σ' is a prefix of the log. According to Definition 4.1, it must be defined a log state s after the sequence σ'. The task t must be in the A_T of s, because σ is a feasible sequence of the model. But t must not appear in R_T because σ is not a prefix of the log. By construction of R_T, t is a Escaping Edge. Therefore, $\sigma \in M$. □

Following with the running example, Fig. 4.11 shows the MDT for the running example model and log. The MDT traces shown are result of the unseen behavior produced by the loop of G: an element of the model that allows to execute as many G as it is wanted, or even to skip without executing G. These behaviors are not reflected in the log, and therefore, there are MDT addressed to them.

MDT
A C G G
A C G D G
A C D G G
A C D H
A C H

Figure 4.11: *Minimal Disconformant Traces for the Running Example. All them are produced by the G loop: a loop that allows to execute as many G as we want, or even to skip the execution of G (these are behaviors not reflected in the log).*

Note that, not all the situations pointed out by some MDT are errors of the model. Some of them may represent meaningful abstractions of the model, in order to have a more general vision of the analyzed process. On the other hand, other MDT traces may represent precision mismatches of a bad-designed or out-of-date model. For that reason, the analysis performed over the set of MDT should be done by an expert in the process domain.

Notice also that MDT are not more than a set of traces, and hence, they can be considered as an event log. Therefore, some of the Process Mining techniques applicable to event logs can be used for MDT too. For instance, the analysis of the tasks frequency to figure out which tasks are the most problematic, or the use of some Process Mining discovery algorithms to derive a model (e.g. a Petri net) that represents all that problematic behavior that makes the model imprecise. A part from the MDT study and analysis itself, MDT are a good starting point for the *Process Refinement*. In other words, the information reflected in the MDT set can be used to correct, update or/and improve the model, being this new model more precise with respect to the log. The Process Refinement topic and the use of MDT for that purpose are explained in detail in Chapter 7 of this thesis.

4.6 Variants, Extensions and Special Cases

In this section we go a step further and we consider the precision conformance for cases different than the general one presented in the previous section. In particular, in Section 4.6.1 we analyze the case where invisible and duplicate tasks are involved in the mapping between Logs and Petri nets. In Section 4.6.2 we consider a variant of the general approach that uses the markings to define the log states (and not the prefixes). In Section 4.6.3 we propose how to deal with non fitting models (models than does not cover the log). Finally, in Section 4.6.4 we consider the extension of the approach for other models rather than Petri nets.

4.6.1 Dealing with Invisible and Duplicate Tasks

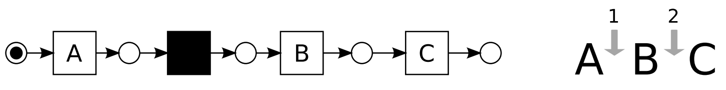
During the presentation of the conformance approach we have been assuming the most simple mapping between log events and petri net transitions, i.e., each event is associated with only one transition, and all transitions has some event associated. However, this is a really strong assumption to make, specially in real scenarios, where is frequent to have more complex mappings, i.e., *invisible tasks* and *duplicate tasks*[2].

In some occasions, the inclusion of invisible or duplicate tasks does not alter the precision analysis methodology presented in previous section. For instance, some of the illustrative examples introduced to present the conformance analysis approach (e.g., the running example) contained invisible or duplicate tasks, but without affecting the analysis process. However, in other occasions the introduction of these kind of tasks makes necessary a revision of the conformance analysis method. In the following sections we illustrate the problems originated by the invisible and duplicate tasks, and we propose techniques to deal with them.

Invisible Tasks

The precision analysis method presented in the chapter of the thesis determines how to associate markings to log states (Sec. 4.2), and how to determine which actions are enable in the current moment (Sec. 4.3), in order to measure the precision of the model. When the mapping between log events and Petri net transitions is simple, determine all this is straightforward.

Actually, it could be also straightforward in some cases involving invisible tasks. This is the case of the example shown in Fig. 4.12. Given the position 1 of the trace in (b), it is easy to see that the only visible action allowed in this point is B (executing the invisible task first). In addition, in position 2, when we try to determine the marking associate to that position replaying in the model the prefix of that state (AB), the presence of an invisible tasks does not affect the result, i.e., there is only one possible way to reach this sequence of tasks.



(a) Model (b) Trace

Figure 4.12: *Example of a non problematic scenario involving Invisible Tasks.*

However, there are cases where the presence of invisible tasks may affect the analysis, making the precision measure unpredictable. This is the case of the example shown in Fig. 4.13. Consider the situation reflected in the figure. Which are the enable visible actions in the position 1 of the trace? Is it only B because there are a *good* sequence of invisibles (Inv2) that only enables C? Or is it B and C because these are the visible tasks enabled considering all the possible sequence of invisible tasks? And if we consider the position 2 of the trace, the thing gets worse. Which sequence of tasks should we fire in order to determine the marking associate to this position? A-Inv1-C or A-Inv2-C? Choosing the first sequence, B will be enabled. However,

[2]The mapping between log events and Petri net transitions has been presented in Sec. 3.3

no tasks will be enabled if we choose A-Inv2-C. Summing up, the introduction of invisible tasks may produce indeterminism. This is a non trivial problem that can only be handled using heuristics and local explorations, producing results not 100% guaranteed from a global point of view.

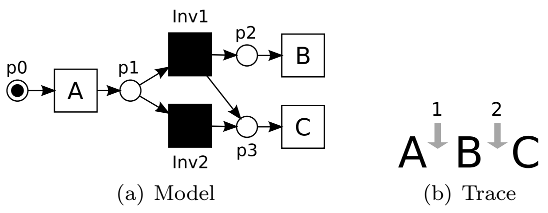


Figure 4.13: *Example of Indeterminism produced by Invisible Tasks. Because invisible transitions have no event associated, we cannot choose determinately which one to fire only examining the log.*

In this thesis we propose the use of the *Invisible Coverability Graph*, in order the to help handling these situations. Invisible Coverability Graph (ICG) is a variant of the classic Coverability Graph (CG) [26], a graph with markings in the vertex, tasks in the edges, that explores approximately the reachability of a Petri net, but avoiding the infinite state space subsuming different finite markings in an infinite $\omega - marking$. The difference between CG and ICG is that ICG only consider invisible tasks as edges of the graph. Notice that, although the use of coverability techniques can be problematic with respect to computational complexity for the general case, is not a strong assumption to consider reasonably small the number of invisible tasks involved. Building the ICG from a given marking, we obtain an approximate exploration of the possible sequence of invisibles and the reached markings. Fig. 4.14 shows the ICG for the example in Fig.4.13.

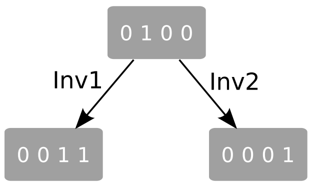


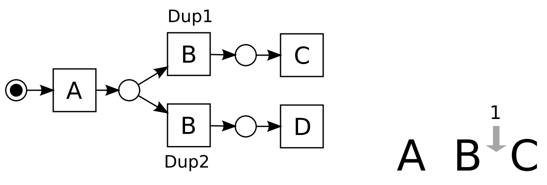
Figure 4.14: *Example of Invisible Coverability Graph. The digits in the vertexes correspond to the markings associated with each vertex, i.e., the number of tokens in p0,p1,p2 and p3, for that marking. The edges of the ICG only include invisible tasks.*

The Invisible Coverability Graph can be use as helping element to apply heuristics and "best effort" strategies in order to try to answer the questions produced by the indeterminism of the invisible tasks. For instance, in order to determine the allowed tasks we can consider the union of all the tasks that are enable in some marking of the ICG (B and C in the example of Fig. 4.13). In addition, in order to choose which sequence of invisibles fire in the prefix replay we can apply different "best effort" strategies. For instance, we can restrict our conformance analysis to cases where there is only one possible marking reached after replaying the prefix. An other example, proposed in [33], could be consider *lazy* the invisible tasks, i.e., firing invisible tasks are only considered if the task to be replayed is not enabled without firing invisibles. In [33] it is also proposed the heuristic of the *shortest sequence of invisibles*, i.e., always choose the shortest sequence of invisibles than enable the task, based on the idea of avoiding the possible side effects between tasks produced by the firing of invisible tasks.

Duplicate Tasks

As the Invisible case, the Duplicate case could be also problematic. This is not always the case. There are situations where the inclusion of duplicate tasks does not disturb the precision analysis. For instance, the running example shown in Fig. 4.4 includes the duplicate task A, but this task does not produce any problem. This is because both transitions of the Petri net associated with A are never enabled at the same time.

However, there are other situations where the presence of duplicate tasks could be a problem. Fig. 4.15 shows an example of these kind of situations. Consider this example. Given the position 1 of the trace, which transition should we choose to replay the prefix AB: Dup1 or Dup2? Again, as in the invisible case, we have a case of indeterminism, that can only be handled with heuristics, e.g., *look ahead* in the log, and use the next tasks in order to choose between duplicate tasks. In this case, given that the next task is C, the transition chosen would be Dup1.



(a) Model (b) Trace

Figure 4.15: *Example of Indeterminism produced by Duplicate Tasks. Because both transitions have the same event associated, we cannot choose determinately which one to fire only examining the log.*

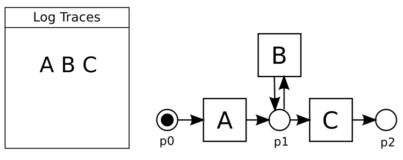
4.6.2 Log States as Markings

In this section we present a variant of the general approach concerning about the derivation of the log states. In the general approach seen so far the log states are defined as *prefixes* of the log (cf. Def. 4.1). Two trace points with different prefix represent two different states. In the variant, this is no longer the case. Instead of that, the element used to identify states are the markings, i.e. two trace points reaching the same marking correspond to the same state. With this approach we reduce the number of log states, and we achieve a more high-level vision of the precision, e.g., several *old* states with the same marking associated are now abstracted as a unique *new* state representing all them.

However, the use of this abstraction produces loss of information. In this particular case we lose the exact moment when the tasks are executed, and therefore losing also accuracy detecting precision problems.

This loss of information can be seen clearly in the example of Figure 4.16. In this example we compare both precision approaches: the general one with prefixes, and the variant with markings as states. Using the general approach we detect two escaping edges, pointing out the extra behavior allowed by the loop and not reflected in the log (skip B or execute more than one B). On the other hand, using the variant approach (where each reached marking define a state), we do not detect any escaping edge. This is because behind a marking we are subsuming several moments, losing the power to distinguish in what moment a task has been executed. For instance, in this example, we are losing all hints about the extra behavior allowed by the loop (and not reflected in the log).

The level of abstraction provided by the variant may be interesting in some cases, e.g., when we want to abstract from the extra behavior allowed by a loop in the model. However, we must



(a) Log (b) Model

State ID	State Prefix	Marking ID	Marking Tokens	Allowed Tasks (A_T)	Reflected Tasks (R_T)	Allowed Tasks (A_T)
1		m0	p0	A	A	
2	A	m1	p1	B C	B	C
3	AB	m1	p1	B C	C	B
4	ABC	m2	p2			

(c) Prefix Approach

State ID	Marking ID	Marking Tokens	Allowed Tasks (A_T)	Reflected Tasks (R_T)	Allowed Tasks (A_T)
1	m0	p0	A	A	
2	m1	p1	B C	B C	
3	m2	p2			

(d) Markings Approach

Figure 4.16: *Comparison between the Prefix Approach (the general approach) and the Markings Approach (the variant). In this variant, each marking reached in the model defines a different state. With this approach we lose information about the exact moment where the actions have been executed because several moments are subsumed in the same marking (the same state). In this example we are losing the information about the extra behavior allowed by the loop but not reflected in the log.*

use this variant carefully, controlling what exact information we are abstracting, and avoiding to lose sight of real precision problems of the model.

4.6.3 Non-fitting Models

In the approach presented so far, we made the assumption that the model was able to replay all the traces in the log, i.e., the fitness value was one. However, this requirement could be difficult to satisfy in some cases, specially in large an real scenarios. The inclusion in the log of non replayable traces may interfere in the precision computation, i.e., the fitness dimension (how much behavior of the observed behavior is captured by the model) is correlated with the precision dimension (how precise is the behavior of the model with respect to the observed behavior in the log). For that reason, similar to [31], we recommend to perform a two phases conformance analysis.

In the first phase the fitness of the system is analyzed, ensuring the replayability of the traces in the log. This analysis of the fitness dimension can be performed using some of the Process Mining techniques. One of the most used technique is the *token game*, presented in [33]. In this method, every trace of the log is tried to be replayed in the log, creating artificial tokens in order to make enable tasks that should to be enable (but they are not). At the end, the artificial tokens and the remaining tokens (indicating a non properly ending of the process execution) are considered mismatches, and they are used to measure the fitness.

Together to this and other fitness methods, we present a new fitness technique based on the basic idea used for the precision approach. Symmetric to the Escaping Edges (used to identify the points where the model deviate from the log), we can define the *Log Escaping Edges* (LE_E), i.e., the points where the log deviates from the model. All this points could be computed, analyzed and evaluated, providing a measure of fitness (instead of precision, in this case). Fig. 4.17 compares the Escaping Edges and Log Escaping Edges.

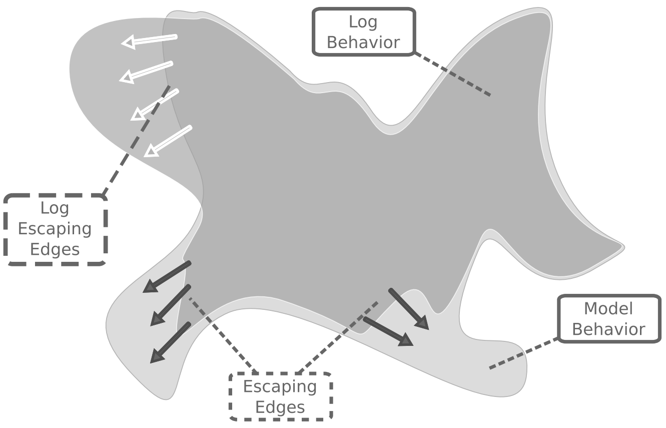


Figure 4.17: *Comparison between Escaping Edges and Log Escaping Edges*

During the second phase, the fitting log is used to compute the precision of the system, using the approach seen in this chapter. Once the analysis is performed, the results may be combined with the fitness results (and other dimensions) to achieve a more complete vision of the system conformance.

4.6.4 Other models

The work of this thesis is based on Petri nets as formal language for modeling. However, the main idea of the approach presented can be generalized to be applied to other workflow modeling languages with a suitable semantic. One possibility may be the *EPC*.

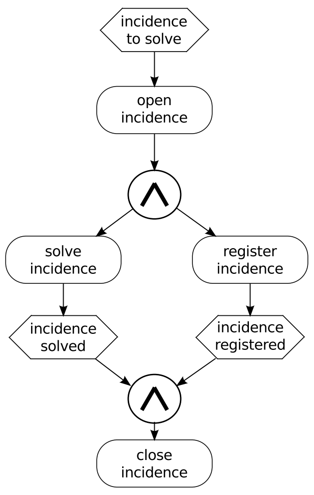
Event-driven Process Chains (EPC) [37] is a graphical process description language, intuitive and easy to understand. The EPC is a widespread process modeling technique because of the success of products such as SAP R/3 and ARIS. EPC uses a *chain* of elements to define the workflow of a process. An event-driven process chain may contain the following elements:

- *Functions*
  A Function is an activity or task which needs to be executed. It is represented as rounded rectangle.

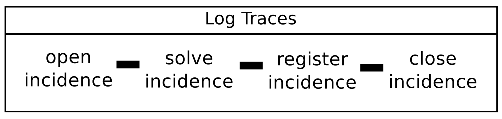
- *Events*
  Events describe the situation before and/or after a function is executed. It is represented as hexagon. In general, an EPC diagram must start with an event and end with an event.

- *Logical Connectors*
  Logical Connectors are used to describe the flow of the process, connecting events and functions. There are three types of connectors: $\vee$ (and), XOR (exclusive or) and $\wedge$ (or).

Figure 4.18 exemplifies the application of the precision conformance approach to processes modeled using Event-driven Process Chains. The process (reflected in the log (b)) represents the management of an incidence: first we open the incidence and then we solve it. Finally, once the incidence is solved, we register it in the data base and we close it. The EPC in (a) models that process. However it does imprecisely: after open an incidence, the model allows to register the incidence, but this behavior is never observed in the log. Therefore, the execution of the register action just after open the incidence may be considered as an *Escaping Edge*, and "open-register" the Minimal Disconformant Trace pointing to it.



(a) EPC



(b) Log

Figure 4.18: *Example of Precision Approach applied to Event-Driven Process Chains (EPC). The model described for the EPC in (a) does not describe precisely the process reflected in the log (b), i.e., after executing the action* open incidence *the model allows to execute the task* register incidence*, but this behavior was never reflected in the log. Therefore, the execution of register just after open the incidence may be considered as an* Escaping Edge*.*

Nevertheless, some times this other modeling languages can be converted or mapped into Petri nets, making them appropriate for using Petri nets analysis methods with them (including the approach presented in this thesis). For example, in [41] the author present a technique to map some Event-driven Process Chains into Petri nets. Some of this conversions between models are implemented in the *ProM* framework [4], one of the most flexible tool in the Process Mining area (it is seen in more detail in the *implementation* chapter of this thesis).

5 Implementation

In this chapter we present the implementation of the precision conformance approach presented in Chapter 4. The approach has been implemented as a *ProM* framework plug-in under the name of *ETConformance*. Therefore, this chapter is divided in two parts: the first one (Sec. 5.1) has the objective of presenting the framework ProM. The second one (Sec. 5.2) presents the plug-in ETConformance itself.

5.1 ProM

ProM [4] is a generic open-source framework for implementing process mining tools in a standard environment. ProM is a project by *the Process Mining Group* [3], and it has been issued under the Common Public License (CPL) license. The tool is composed by plug-ins, each one implementing a different approach of the Process Mining area. There are more than 230 plug-ins such as:

- ***Discovery Plug-ins***
  Some of the most important Process Mining discovery miners are implemented in ProM. For instance, there are plug-ins implementing the *Alpha algorithm* [46], the *Genetic Miner* [14], the *Heuristic Miner* [52], *Fuzzy Miner* [19], or the *Parikh Miner* [47]. This wide variety of miners makes interesting the implementation in ProM of our conformance approach, in order to compare the quality of the models mined using different techniques. A part from control-flow perspective, ProM also includes plug-ins for the other perspectives, such as the *Social Network Miner* [43] or the *Organizational Miner*.

- ***Analysis Plug-ins***
  ProM includes several plug-ins for analyzing and verifying models and logs. This category includes the Process Mining Conformance approaches. The implementation of our approach in ProM makes possible to compare its conformance results with other conformance approaches such as *Conformance Checker* [33] or the *Minimal Description Length* analysis plug-in [6]. ProM also includes other analysis plug-in such as *Performance Analysis with Petri Net* or the *Dotted Chart Analysis* plug-in.

- ***Input Output Plug-ins***
  One of the strong points of ProM is the possibility to work with different modeling languages. There are plug-ins to deal with Petri nets, EPCs, YAWLs, BPEL, etc. There

are also plug-ins for converting between formats, e.g. from Petri nets to YAWLs, or from YAWL to ETC. This opens the door to input a model in some format, convert it to Petri nets, and then apply the approach proposed in this thesis.

- ***Other Plug-ins***
  Finally, ProM also includes other plug-ins such as log filters, or plug-ins to support the step-by-step redesign of process models.

Recently, a new version of ProM has been developed. The release of this new version, called ProM 6, is expected in the *Bussiness Process Management* (BPM) conference on September 2010.

5.2 ETConformance Plug-in

The conformance approach presented in this thesis has been implemented as a ProM 6 plug-in. This plug-in is called *ETConformance*, and it is included in the official plug-in repository of ProM.

The use of ETConformance is really intuitive. First, an Event Log and a Petri Net must be imported to ProM. This can be done using the *Import* tab. Once the two main elements of the conformance analysis are been imported, we use them as inputs of the ETConformance plug-in. The plug-in requires also the initial marking of the Petri net (cf Fig. 5.1).

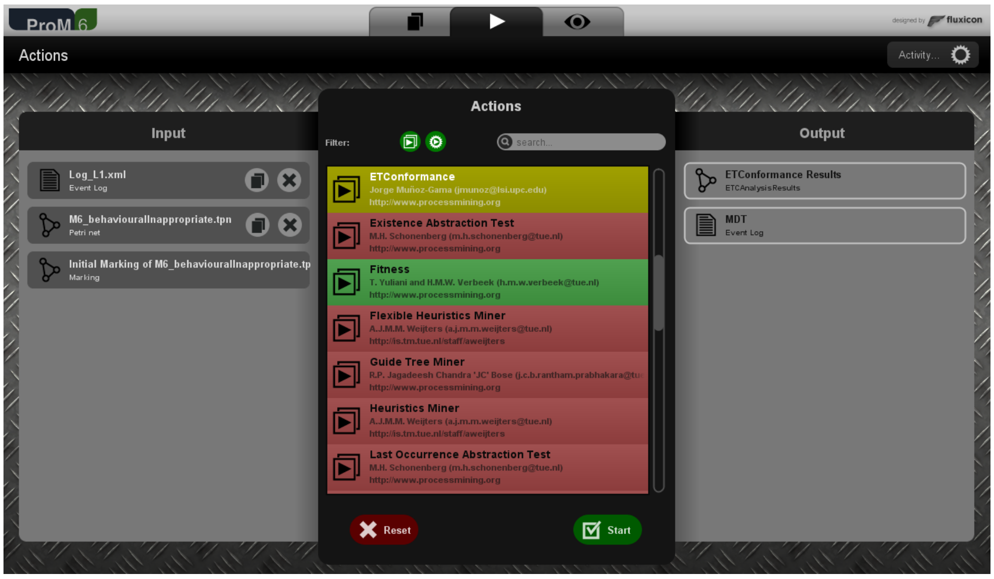
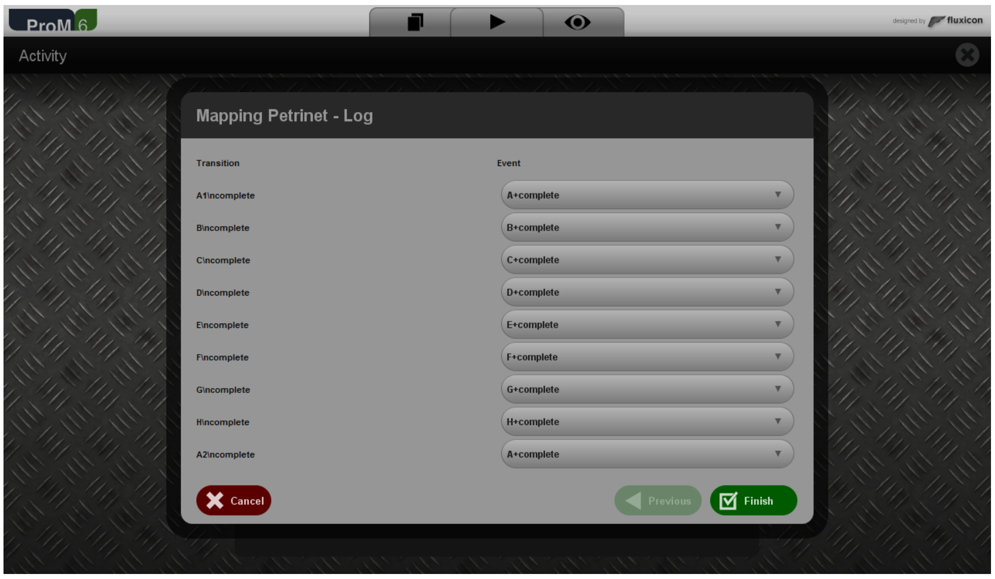

ProM 6
designed by fluxicon
Actions
Activity...
Input
Log_L1.xml
Event Log
M6_behaviourallnappropriate.tpn
Petri net
Initial Marking of M6_behaviourallnappropriate.tp
Marking
Actions
Filter:
search...
ETConformance
Jorge Muñoz-Gama (jmunoz@lsi.upc.edu)
http://www.processmining.org
Existence Abstraction Test
M.H. Schonenberg (m.h.schonenberg@tue.nl)
http://www.processmining.org
Fitness
T. Yuliani and H.M.W. Verbeek (h.m.w.verbeek@tue.nl)
http://www.processmining.org
Flexible Heuristics Miner
A.J.M.M. Weijters (a.j.m.m.weijters@tue.nl)
http://is.tm.tue.nl/staff/aweijters
Guide Tree Miner
R.P. Jagadeesh Chandra 'JC' Bose (j.c.b.rantham.prabhakara@tu
http://www.processmining.org
Heuristics Miner
A.J.M.M. Weijters (a.j.m.m.weijters@tue.nl)
http://is.tm.tue.nl/staff/aweijters
Last Occurrence Abstraction Test
M.H. Schonenberg (m.h.schonenberg@tue.nl)
http://www.processmining.org
Reset
Start
Output
ETConformance Results
ETCAnalysisResults
MDT
Event Log

Figure 5.1: *Initial Dialog of ETConformance plug-in. The plug-in requires a log, and a Petri net with its initial marking. It outputs the conformance analysis results, and the set of Minimal Disconformant Traces (MDT).*

Before start, the ETConformance check the existence of a mapping between the Event Log and the Petri Net. If this is not the case, the ProM will show a dialog to create this mapping, i.e., selecting the event associated to each transition of the Petri net (cf Fig. 5.2). On the left we will have the list of the Petri Net Transitions, and on the right we could select the Log Event corresponding to it (or *INVISIBLE* if the transition is not reflected in the log)

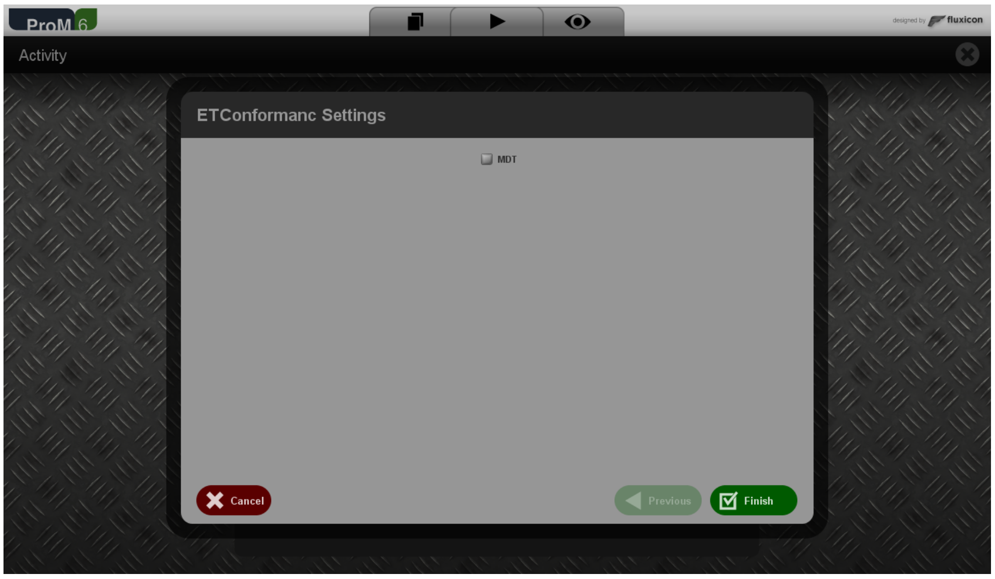


Mapping Petrinet - Log

Transition	Event
A1\ncomplete	A+complete
B\ncomplete	B+complete
C\ncomplete	C+complete
D\ncomplete	D+complete
E\ncomplete	E+complete
F\ncomplete	F+complete
G\ncomplete	G+complete
H\ncomplete	H+complete
A2\ncomplete	A+complete

Figure 5.2: *Mapping Dialog of ETConformance. This dialog helps to associate events in the event log with transitions in the Petri net.*

Once the mapping has been done, ETConformance will ask if we want to compute the set of Minimal Disconformant Traces (MDT) or not (cf. Fig. 5.3). If we uncheck the box, the analysis will consume less time and memory, but we will have only the number of MDTs and not the MDT traces themselves.



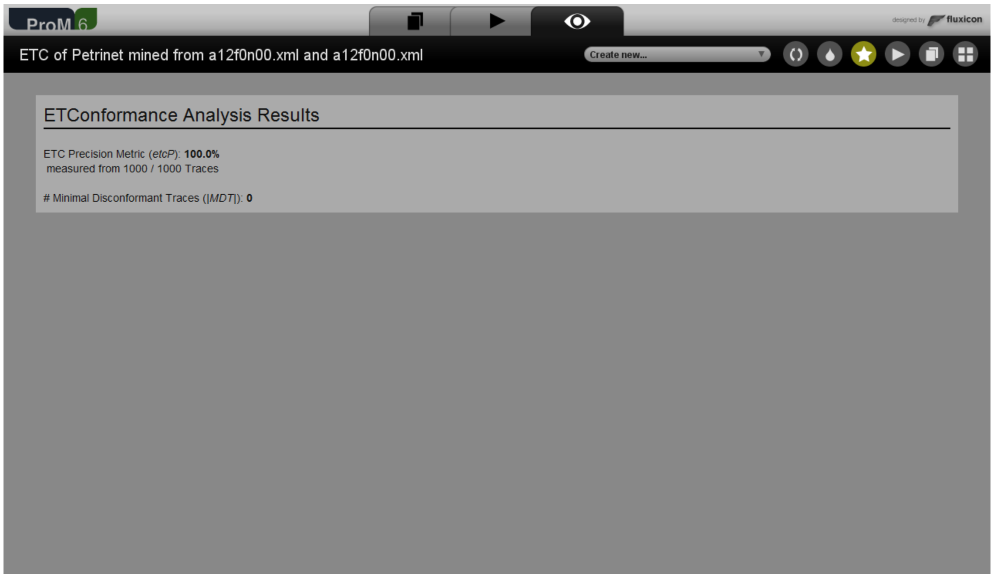
ETConformanc Settings

☐ MDT

Figure 5.3: *Option Dialog of ETConformance. This dialog is used to choose the settings of the plug-in. In the current version, the only option available is too compute or not the MDT set.*

After finishing the analysis, the ETConformance plug-in will output the results (cf. Fig. 5.4).

These results include the value of the ETC Precision Metric (etc_P), and the number of traces of the log used to compute the metric (only the ones that can be replayed in the Petri Net). Moreover, the results include the number of existing MDTs.



ETC of Petrinet mined from a12f0n00.xml and a12f0n00.xml

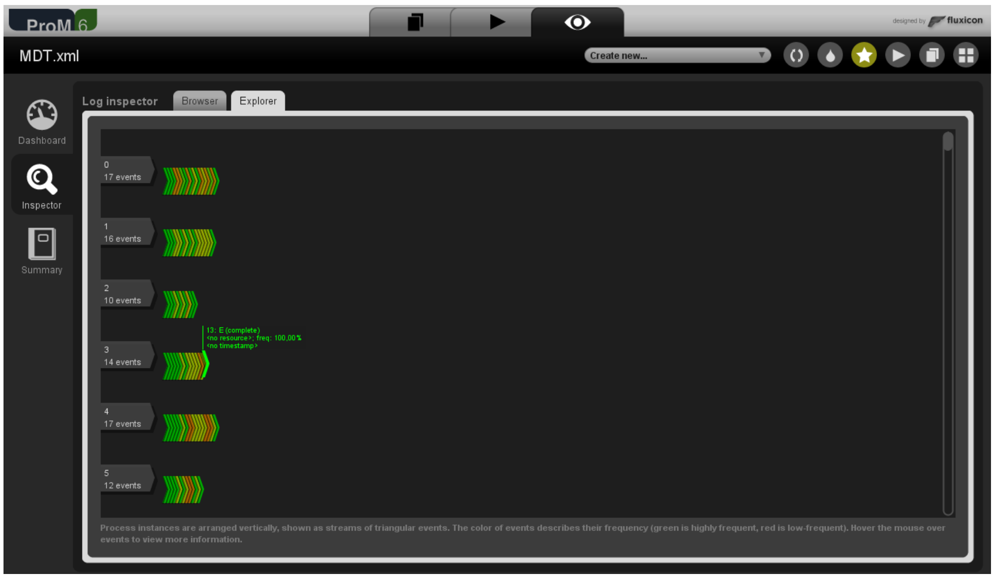
ETConformance Analysis Results

ETC Precision Metric (*etcP*): **100.0%**
measured from 1000 / 1000 Traces

Minimal Disconformant Traces (|*MDT*|): **0**

Figure 5.4: *Results of ETConformance. This results screen includes the value for the* etc_P *metric, the number of traces used in the precision analysis, and the number of MDT.*

Finally, if the MDT check box has been checked, the plug-in will output a log including all the MDT traces (cf. Fig. 5.5). The format of this log is the same as the format used for Event Logs in ProM, and therefore, it can be used as input for other ProM plug-ins.



13: E (complete)
<no resource>; freq: 100,00%
<no timestamp>

Process instances are arranged vertically, shown as streams of triangular events. The color of events describes their frequency (green is highly frequent, red is low-frequent). Hover the mouse over events to view more information.

Figure 5.5: *Inspection of MDT. The format of the MDT is the same as the rest of logs in ProM. Therefore, the same tools to inspect the logs in ProM can be used to analyze the MDTs.*

6 Experimental Results

The precision conformance technique presented in Chapter 4, implemented as the *ETConformance* plug-in within ProM 6, has been evaluated on existing public-domain benchmarks [3]. The purpose of the experiments is:

- Justify the existence of a new metric to evaluate precision, i.e. demonstrate the novelty of the concept when compared to previous approaches.
- Show the capacity of the technique to handle large specifications.

Table 6.1 shows a comparison of the technique presented in this thesis with the technique presented in [33], implemented in ProM 5.2 as the *Conformance Checker*. The rows in the table represent benchmarks with small size (few traces and small Petri net). The names are shortened, e.g., GFA5 represents GroupedFollowsA5. We report the results of checking precision for both conformance checkers in columns under a'_B and etc_P, respectively, for the small Petri nets obtained by the ILPMiner [47] which derived Petri nets with fitness value one. For the case of our checker, we additionally provide the number of minimal disconformant traces ($|\mathsf{MDT}|$). We do not report CPU times since checking precision in both approaches took less than one second for each benchmark.

From Table 6.1 one can see that when the model describes precisely the log, both metrics provide the maximum value. Moreover, when the model is not a precise description of the log, only three benchmarks provide opposite results (GFBN2, GFl2l, GFl2lSkip). For instance, for the GFl2lSkip benchmark, a'_B is providing a significant lower value: this is because the model contains an optional loop that is always traversed in the log. This variability is highly penalized by simply observing the tasks relations. On the other hand, metric etc_P will only penalize the few situations where the escaping edges appear in the log.

Larger benchmarks for which *Conformance Checker* cannot handle are provided in Table 6.2. For these benchmarks, we report the results (precision value, number of MDT and CPU time in seconds) for the models obtained by the ILPMiner [47] and the RBMiner [39]. These are two miner that guarantee fitness value one. For each one of the aN benchmarks, N represents the number of tasks in the log, while the _1 and _5 suffixes denote its size: 100 and 900 traces, respectively. The $t32$ has 200 (_1) and 1800 (_5) traces. The pair of CPU times reported denote the computation of etc_P without or with the collection of MDT (in parenthesis). Also, we provide the results of the most permissive models, i.e., models with only the transitions but without arcs or places (M_T). These models allow any behavior and thus, they have a low etc_P value, as expected.

Benchmark	a'_B	etc$_P$	|MDT|
GFA6NTC	1.00	1.00	0
GFA7	1.00	1.00	0
GFA8	1.00	1.00	0
GFA12	1.00	1.00	0
GFChoice	1.00	1.00	0
GFBN1	1.00	1.00	0
GFParallel5	1.00	0.99	11
GFAL1	1.00	0.88	251

Benchmark	a'_B	etc$_P$	|MDT|
GFl2lOpt	1.00	0.85	7
GFAL2	0.86	0.90	391
GFDrivers	0.78	0.89	2
GFBN3	0.71	0.88	181
GFBN2	0.59	0.96	19
GFA5	0.50	0.57	35
GFl2l	0.47	0.75	11
GFl2lSkip	0.30	0.74	10

Table 6.1: *Comparative table between a'_B and etc$_P$ for small benchmarks.*

		M_T	ILPMiner					RBMiner				
Benchmark	|TS|	etc$_P$	$\|P\|$	$\|T\|$	etc$_P$	|MDT|	CPU	$\|P\|$	$\|T\|$	etc$_P$	|MDT|	CPU
a22f0n00_1	1309	0.06	19	22	0.63	1490	0(0)	19	22	0.63	1490	0(0)
a22f0n00_5	9867	0.07	19	22	0.73	9654	0(3)	19	22	0.73	9654	0(4)
a32f0n00_1	2011	0.04	31	32	0.52	2945	0(0)	32	32	0.52	2944	0(1)
a32f0n00_5	16921	0.05	31	32	0.59	22750	2(10)	31	32	0.59	22750	2(11)
a42f0n00_1	2865	0.03	44	42	0.35	7761	0(2)	52	42	0.37	7228	0(2)
a42f0n00_5	24366	0.04	44	42	0.42	60042	5(28)	46	42	0.42	60040	6(29)
t32f0n00_1	7717	0.03	30	33	0.37	15064	1(15)	31	33	0.37	15062	1(12)
t32f0n00_5	64829	0.04	30	33	0.39	125429	9(154)	30	33	0.39	125429	8(160)

Table 6.2: *Experimental Results Table for etc$_P$ with Large Benchmarks.*

A first conclusion on Table 6.2 is the capability of handling large benchmarks in reasonable CPU time, even for the prototype implementation carried out. A second conclusion is the loss of precision of the metric with respect to the increase of abstraction in the mined models: as soon as the number of tasks increases, the miners tend to derive models less precise to account for the complex relations between different tasks. Often, these miners derive models with a high degree of concurrency, thus accepting a potentially exponential number of traces which might not correspond to the real number of traces in the log.

Finally, three charts are provided: the relation between the log size with respect to the CPU time, the etc$_P$ value and the number of MDTs are shown in Fig. 6.1. For these charts, we selected different log sizes for different types of benchmarks (a22f0, a22f5, a32f0,a32f5 for the two bottom charts, a42f0, t32f5 and t32f9 for the top chart). For the two bottom charts, we used the Petri nets derived by the ILPMiner to perform the conformance analysis on each log, whereas we use a single Petri net for the top chart to evaluate the CPU time (without collecting MDTs) on different logs, illustrating the linear dependence of our technique on the log size. The chart on top clearly shows the linear relation between log size and CPU time for these experiments, which is expected by the technique presented in Chapter 4. The two charts on bottom of the figure show: in the left one, since for the a22/a32 benchmarks the models derived are very similar independently of the log, the more traces are included more traces describing the common behavior are found. On the other hand, the inclusion of more traces contributes to the incorporation of more MDTs, as it is shown in the right chart at the bottom.

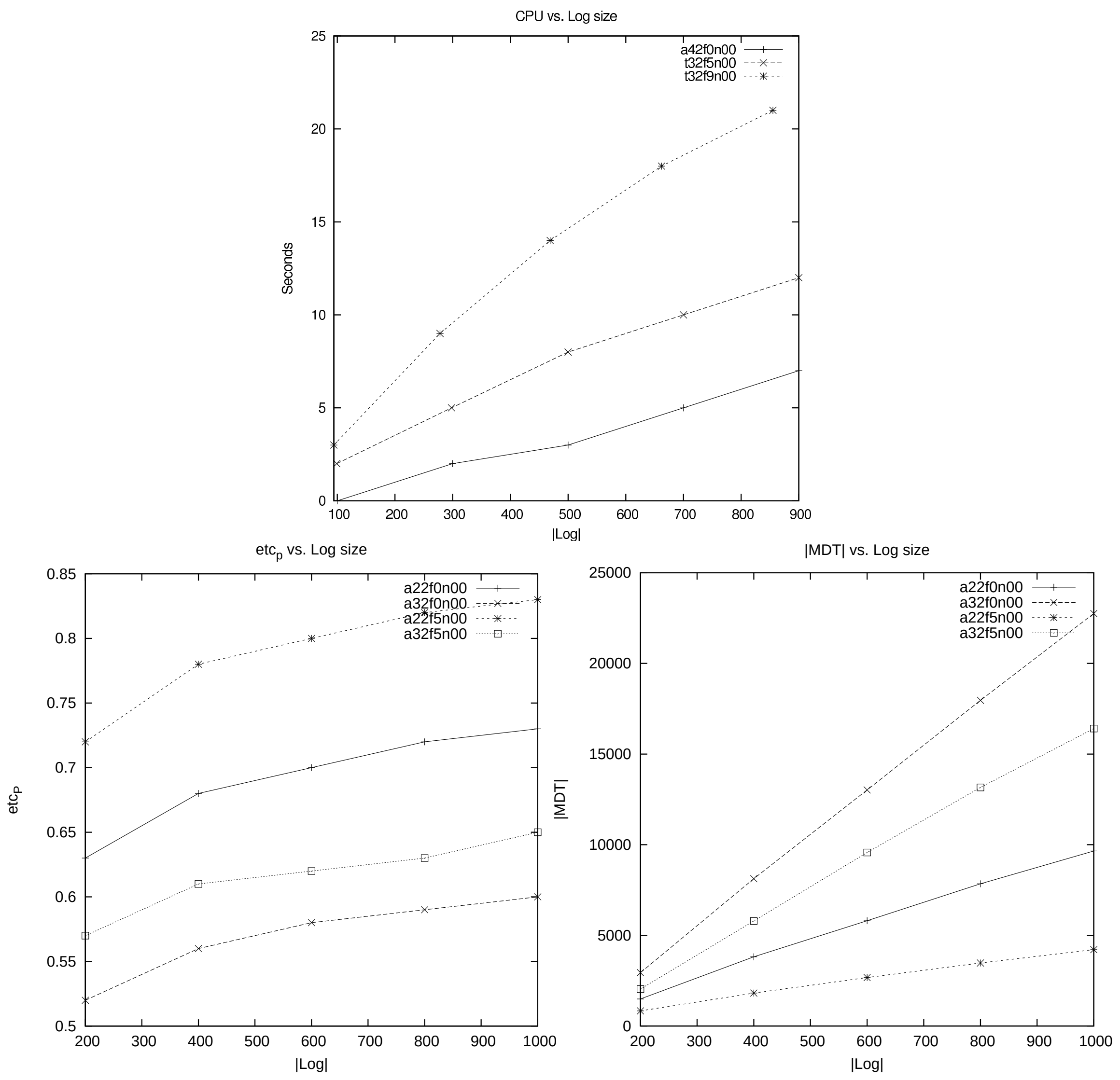


Figure 6.1: *CPU, etc$_P$ and* MDT *versus Log Size for Large Benchmarks.*

7 Process Refinement Approach

Checking the correctness between two objects has always associated a *point of view*, i.e., which of the two elements is the correct one, being the other element the one containing all the inconsistencies. In Process Mining, the precision checking, and conformance analysis in general, is not an exception. Depending on which point of view we choose (log or model), we distinguish between *model perspective* and *log perspective*.

In *model perspective*, the model is considered to be correct, and therefore, all process must satisfy this model. The model may be developed by experts in the area (e.g., banks, shop management, . . .), knowing by their expertise how the process should be. Or it could be also the result of laws and regulations, concerning about the requirements that the processes in that area must satisfy (e.g., the model describing the driving license application process, whose requirements, periods and terms are regulated by the law). In all these cases, the model is considered to be *prescriptive*, and any conformance inconsistence found must be corrected changing how the processes are executed in the real world, in order to adjust them to the model.

On the other hand, we can consider the *log perspective* as point of view. In this case, the log is considered to be correct because it is the direct reflex of how the process are being executed. Hence, the model is then used in a *descriptive* manner, i.e., it describes the real world and its processes.

The use of models reflecting the reality is a powerful tool for the people in charge to make the decisions in the system. It gives a schematic vision of the process, providing also a mechanism to analyze the processes, detecting possible problems and bottlenecks, and giving the opportunity to see how the changes could affect the system. However, all these advantages have a strong dependence in the correctness of the model in the moment of taking the decisions. The use of an incorrect model (or a model out-of-date not reflecting the current reality) to manage and control the business processes could have catastrophic consequences in terms of time, money and even juridical responsibilities.

If a model is incorrect or out-of-date, we can discard it, and derive a new model (like the one that takes an other *picture* of the reality). However, this decision may suppose a huge cost, especially if the model has to be create manually by experts in the area. Even using some of the Process Mining discovery techniques, the results may be not accurate to the reality. Hence, instead of creating a new model from scratch, it could be more interesting to correct the old one, making it a more accurate representation of the reality, i.e., *Refinement*.

This chapter of the thesis is based in this process refinement. In particular, we focus on the use of the precision conformance results (cf. Minimal Disconformant Traces) in order to restrict the model, increasing the precision and obtaining a more accurate model. In the following sections we propose two techniques for precision refinement called *Breaking concurrencies* and *Supervisory Control Refinement*.

7.1 Breaking Concurrencies

As it was explained in the previous section, the aim of precision refinement is to refine the model, restricting its behavior, and correcting its precision discrepancies. There are a wide variety of causes for these discrepancies, and they change from one scenario to another. However, it exists a common situation that repeats in several scenarios and causes imprecisions: the *concurrencies*.

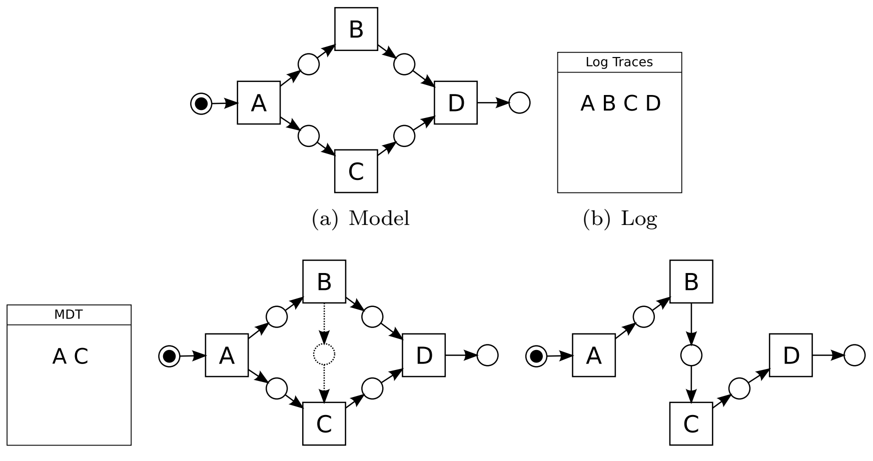
The concurrency between tasks describes the situation where a set of tasks are enabled, and can be executed in any order. If this situation is reflected in the model but not in the log, it produces an imprecision, i.e., the model describes more behavior than the one reflected in the log. The main idea of the *breaking concurrencies* technique proposed in this section is to restrict this extra behavior introducing a new *constraint* in the model. In our case, being Petri nets the modeling language chosen, this constraint is a new *place* between concurrent tasks in the model, breaking this concurrency and making the tasks being sequential, reflecting more accurately the behavior of the log.

Simple Motivational Example
In Fig. 7.1 we show a simple example to motivate and illustrate the general idea of the approach presented in this section. This figure contains the initial model (Fig. (a)) and event log (Fig. (b)) of a system. After performing the precision conformance analysis we detect that this model does not reflect precisely the behavior of the log, i.e., the etc_P metric in this system is 0.8 (different than one), and therefore the Minimal Disconformant Traces set is not empty (shown in Fig. (c)). After analyzing the Petri net we detect that it models a concurrent behavior between B and C. However, analyzing the event log we detect a sequential behavior between B and C, being that the cause of the precision discrepancy. To correct this imprecision, we introduce a new place in the model between tasks B and C, making them sequential (Fig. (d)). After checking the precision and fitness of this new model we realize that it describes completely precisely the log without losing its fitness (i.e., all traces accepted by the old model are accepted by the new model too). Finally, we remove the redundant places (i.e., removing them does not modify the behavior), simplifying the structure of the model (Fig. (e)).

In the following sections we will present in detail each part of the approach: in Sec. 7.1.1 we present the methods to obtain the concurrency relations from Petri Nets. In Sec. 7.1.2 we do the same for the event log. Finally, in Sec. 7.1.3 we propose the complete algorithm to correct precision discrepancies using the precision analysis results and the concurrency information. For the technique presented in this section we assume the most general Petri net / Event log mapping, focusing on the cases without invisible or duplicate tasks.

7.1.1 Model Concurrencies

In a Petri net, two transitions are considered concurrent if they can occur concurrently from some reachable marking of the net, i.e., if there is a reachable marking that enables both transitions and firing one does not disable the execution of the other. A formalization of this concept can be found in Def. 7.1.

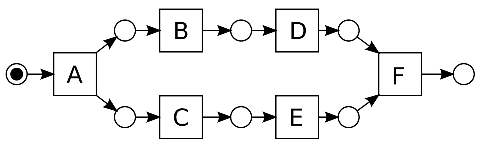


(a) Model (b) Log

(c) MDT (d) Refined Model (e) Final Model

Figure 7.1: *Motivational Example for the Breaking Concurrencies Approach. The example is composed by a model (a), a log (b), and the Minimal Disconformant Traces (c) resulting from checking the precision between them . The model resulting after breaking the concurrency between B and C is shown in (d). In (e) the final refined model after removing the redundant places is presented.*

Definition 7.1 (Concurrency relation (for transitions)) *Let* $\mathsf{PN} = (P, T, W, m_0)$ *be a Petri net. Given* $t \in T$*, we define the marking* m_t *of* PN *as the marking that puts* $W(p, t)$ *tokens for every input place* p *of* t*, and no tokens elsewhere. The* concurrency relation *(restricted to transitions)* $\| \subseteq T \times T$ *contains the pairs* (t_1, t_2) *such that* $m \geq m_{t_1} + m_{t_2}$ *for some marking* $m \in [m_0\rangle$*.*



(a)

B	$\|$	C
B	$\|$	E
C	$\|$	D
D	$\|$	E

(b)

Figure 7.2: *Example of Model Concurrencies. The model analyzed is shown in (a), and the list of relations is shown in (b).*

Fig. 7.2(b) shows the concurrent relations for the model in 7.2(a). Computing the relations for this simple example is easy, but in the general case the cost of computing this relation is huge, needing the complete exploration of the state space, even existing the possibility of having infinite reachable markings in case of unbounded Petri nets. For that reason we propose an alternative best effort strategy based on the *structural concurrency relation*, first presented in [21]. The idea is to use an alternative relation (the structural concurrency relation in this case) that overapproximates the concurrency relation, but that can be computed in polynomial time.

In order to present the structural concurrency relation, first we need to generalize the definition of concurrency relation seen in Def. 7.1, to extend it for concurrency between nodes (i.e., Transitions and Places) of the Petri net.

Definition 7.2 (Concurrency relation (for nodes)) *Let* $\mathsf{PN} = (P, T, W, m_0)$ *be a Petri net, and let* $X = P \cup T$ *be the set of nodes. Given* $x \in X$*, we define the marking* m_x *of* PN *as: (i) if* x *is a place, then* m_x *is the marking that puts one token on* x *and no tokens elsewhere; (ii) if* x *is a transition, then* m_x *is the marking that puts tokens for every input place* p *of* t*, and no tokens elsewhere. The* concurrency relation $\| \subseteq X \times X$ *contains the pairs* (x_1, x_2) *such that* $m \geq m_{x_1} + m_{x_2}$ *for some marking* $m \in [m_0\rangle$*.*

Definition 7.3 (Structural concurrency relation [21]) *Let* $\mathsf{PN} = (P, T, W, m_0)$ *be a Petri net, and let* $X = P \cup T$ *be the set of nodes. The* structural concurrency relation $\|^S \subseteq X \times X$ *is the smallest symmetric relation such that:*

i. for all places p, p'*, if* $m_0 \geq m_p + m_{p'}$*, then* $(p, p') \in \|^S$

ii. for all transitions t*, if* $({}^\bullet t \times {}^\bullet t) \setminus id_P \subseteq \|^S$*, then* $(t^\bullet \times t^\bullet) \setminus id_P \subseteq \|^S$*, where* id_P *denotes the identity relation on the places of* P*.*

iii. for all nodes x *and for all transitions* t*, if* $\{x\} \times {}^\bullet t \subseteq \|^S$*, then* $(x, t) \in \|^S$ *and* $\{x\} \times t^\bullet \subseteq \|^S$*.*

The next step is to prove that Structural concurrency relation is an overapproximation of the concurrency relation. This is done in Theorem 7.1, where it is claimed that the structural concurrency relation is a subset of the concurrency relation (in the specific Petri net class of *live and bounded free-choice STGs*, both sets are even equal). The proof of the theorem can be found in [22].

Theorem 7.1 ([22]) *For general Petri nets,* $\| \subseteq \|^S$*. For live and bounded free-choice STGs,* $\| = \|^S$*.*

In contrast to concurrency relation case, there is a polynomial algorithm to compute the structural concurrency relation of a Petri Net (Algorithm 3). This algorithm corresponds to the one presented in [16] (a corrected version of the algorithm previously presented in [22]). If subset $R \subseteq X \times X$ is encoded as a two dimensional bitarray, the algorithm needs $O(|X|^2)$ space and $O(|P|^2 \cdot |T| \cdot |X|)$ time[1].

7.1.2 Log Concurrencies

The second part of the procedure of breaking concurrencies is to detect concurrencies in the log. Actually, derive the concurrency relations of the log is not the main goal of this part of the procedure. Instead, we are interested in detecting the *lack* of concurrency between tasks, i.e., this absence of concurrency in the log together with the presence of concurrency in the model points out a precision problem in the system, where the model allows more behavior that the one reflected in the log.

In the literature exists different approaches to derive event relations from an event log. In [46], the authors present some log-based ordering relations, such as $a \rightarrow b$ (referred as *direct causal relation*, $a \parallel b$ (suggesting *potential parallelism*), or $a\#b$ (indicating *possible conflict*). In [53] this list of ordering relations is modified and extended to include some other relations such as $a \triangleleft b$ (corresponding to XOR-Split) and $a \triangleright b$ (corresponding to XOR-Join). Alternatively, in [8], the authors present a technique for detecting concurrency information based on probabilistic analysis of the event traces.

[1]It is possible to give a faster algorithm for free-choice models. This new algorithm runs in $O(|X|^2)$ space and $O(|P| \cdot |X|^2)$ time, and can be seen in [16] too.

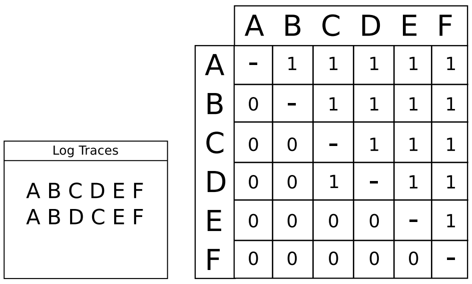
```
Input: Petri Net: $\mathsf{PN} = (P, T, W, m_0)$
Output: Structural Concurrency Relation: $R \subseteq X \times X$
$R := \{(p, p') \mid m_0 \geq m_p + m_{p'}\} \cup (\bigcup_{t \in T} (t^\bullet \times t^\bullet) \setminus \mathrm{id}_P)$;
$E := R$;
while $E \neq \emptyset$ do
    select$(x, p) \in E$; $E := E \setminus \{(x, p)\}$;
    for every $t \in p^\bullet$ do
        if $\{x\} \times {}^\bullet t \subseteq R$ then
            $\mathrm{Aux} := (\{x\} \times t^\bullet) \cup (t^\bullet \times \{x\}) \cup \{(x, t), (t, x)\}$ ;
            $E := E \cup ((\mathrm{Aux} \cap (X \times P)) \setminus R)$;
            $R := R \cup \mathrm{Aux}$ ;
        end
    end
end
```

Algorithm 3: Structural Concurrency Relation Algorithm (Kovalyov & Esparza) [16]

The approach proposed in this section is based on the *Firing Causalities* relation. This relation not only provides us information about the lack of concurrency, but also the information needed to determine the number of tokens of the new place introduced to break the model concurrency (this is explained in detail latter in this section). In order to present the concept of Firing Causalities, first we need to introduce the concept of *Firing Causalities Matrix.*

Definition 7.4 (Firing Causality Matrix) *Let* EL *be an event log, n the number of tasks in* EL*, and Pref*(EL) *the set of all prefixes for traces in* EL*. Let $\#(\sigma, t)$ be the number of occurrences of task t in the trace σ. We define the* Firing Causality Matrix $M \in \mathbb{N}^{n \times n}$ *such that* $M(i, j) = max\{\#(\sigma, t_i) - \#(\sigma, t_j) \mid \sigma \in Pref(\mathsf{EL})\}$.

Definition 7.5 (Firing Causality) *Let M be the Firing Causality Matrix of the event log* EL*. There is a* firing causality *between task t_i and t_j if $M(i, j) > 0$ and $M(j, i) = 0$.*


Log Traces

A B C D E F
A B D C E F

	A	B	C	D	E	F
A	-	1	1	1	1	1
B	0	-	1	1	1	1
C	0	0	-	1	1	1
D	0	0	1	-	1	1
E	0	0	0	0	-	1
F	0	0	0	0	0	-

(a) Log (b) Firing Causalities

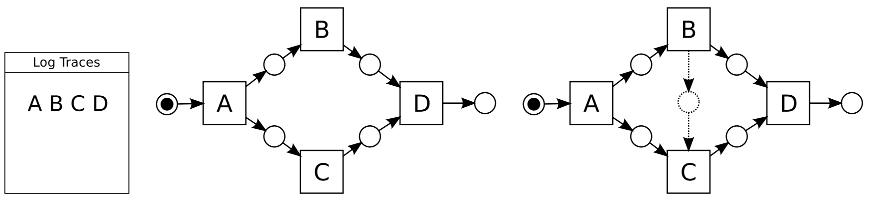
Figure 7.3: *Example of Firing Causality Matrix. The figure shows the Firing Causality Matrix (b) for the log in (a).*

Fig. 7.3(b) shows the Firing Causality Matrix of the log in (a). The use of the firing causalities of the log as a guide for adding new places to the model is *fitness preserving* with respect to the log, i.e., any trace in the log able to be replayed in the original model, can still be reproduce in the new model. This is claimed and proved in Lemma 7.1.

Lemma 7.1 *Let* EL *and* PN *be an Event Log and a Petri net respectively. Let* PN′ *be the same Petri net with a new empty place from t_o to t_d. If there is a firing causality between t_o and t_d, then the fitness of* PN *with respect to* EL *is the same as the fitness of* PN*' with respect to* EL.

Proof: We assume the existence of a trace σ reproducible in PN but not in PN'. Given that the only new restriction added to PN' with respect to PN is the new empty input place p for t_d, it is straightforward to see that the problem can only come from the impossibility of firing t_d during the replay of σ because this new place. In other words, there is a moment m during the replay of σ where t_d should be fired, but p has no tokens in that moment, making t_d not enabled. Let σ_m the sequence of fired events of σ until the moment m. Given that the only task that puts tokens in p is t_o and the only one that removes tokens is t_d, for being p empty in the moment m, the number of occurrences of t_o in σ_m must be lower or equal than the number of occurrences of t_o in σ_m. However, by definition of firing causality between t_o and t_d, this situation is impossible. In conclusion, the existence of σ is a contradiction. □

On the other hand, the addition of a new place guided by the firing causalities may result in a increase of the precision, making the model a more accurate reflection of the log behavior. This is because, with the new place between a task t_1 and a task t_2, we are disabling t_2 until t_1 is executed, limiting the behavior of the model, and being more according with the log behavior. The motivational example shown in Fig. 7.4 illustrates clearly this precision improvement, i.e., while the model in (b) has some fitness issues, the model in (c) describes the process with complete precision.



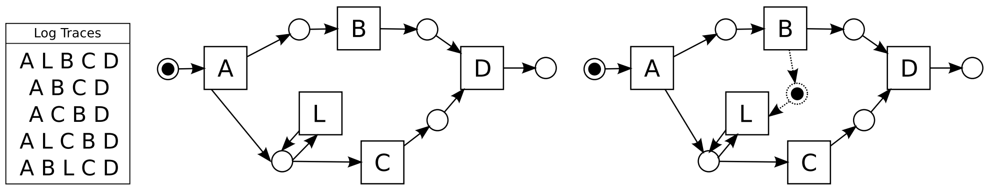
(a) Log (b) Model (c) Refined Model

Figure 7.4: *Example of Breaking Concurrencies Precision Improvement. The refined model (c) describes more precisely the log (a) than (b) without losing fitness.*

Notice that, we can go a step further and use the Firing Causalities and the Firing Causalities Matrix in a more general way in order to introduce places. In particular, including tokens for the new places in the initial marking of the model, we can break concurrencies between tasks without a *pure* firing causality relation. Fig. 7.5 illustrates this case. In this simple example, according to Definition 7.5, there is no firing causality between B and L, because $M(B, L) = 1 > 0$ but $M(L, B) = 1 \neq 0$. However, adding a new place between B and L, and setting to 1 (according to $M(L, B)$) the number of tokens in the initial marking for this new place, we can increase the precision of the system from 0.90 to 0.95, without losing fitness.

7.1.3 Breaking Concurrency Approach

In the previous sections we have presented some possible methods to obtain relation information from Event Logs and Petri Nets. In this section we propose an algorithm that uses this



(a) Log (b) Model (c) Refined Model

Figure 7.5: *Example of Breaking Concurrencies with Initial Tokens. In this example, adding a new place with 1 initial token (c) we improve the precision of (a) and (b), without losing fitness.*

information to refine a given Petri Net.

```
Input: EL: Event Log ; PN: Petri Net
$C_f$ := computeConformance (EL,PN) ;
$R_l$ := computeLogRelations (EL) ;
while finishConditionReached () do
    $R_m$ := computeModelRelations (PN) ;
    PN' := breakConcurrency (PN,$R_l$,$R_m$,$C_f$) ;
    $C_f$ := computeConformance (EL,PN') ;
    PN := selectModel (PN',PN,$C_f$) ;
end
postProcessing (PN) ;
```

Algorithm 4: Breaking Concurrencies Algorithm

computeConformance
At the beginning of the algorithm, the conformance between the original model and the event log is checked in order to have a starting point for the model refinement. This conformance checking includes fitness and precision measurements, together with other conformance elements such as the Minimal Disconformant Traces (cf. Sec. 4.5). The second time *computeConformance* appears in the algorithm, is for checking the conformance of the new model.

computeLogRelations
The computation of the log relations is done only once, at the beginning of the algorithm, because the log does not change in all the refinement process. The procedures for detecting tasks relations from an event log have been explained in the Section 7.1.2.

computeModelRelations
Unlikely log relations, model relations must be computed each time the model is modified. If the model is small enough, the exact concurrency relation can be computed. However, for large models the structural concurrency relation should be computed. Both relations has been explained in Section 7.1.1.

breakConcurrency
This is the function that choose where to put the new place, in order to *break the concurrency.* This decision is made according to the log and model relations, but also according to the information provided by the conformance results and the other conformance

elements (e.g., prioritize a place that *removes* some of the most frequent minimal disconformant traces). This function also checks that the chosen place is valid (e.g., there is no other place with the same input and output transitions).

selectModel
The objective of this function is to choose between the new model (with the new place) or the original one. The main criterion for selection one model or the other is the precision conformance result, i.e., it makes no sense to choose a more complex model if the precision has not change. Notice that, because the model relations are an overapproximation, we do not have any guarantee about the adequacy of the place inserted. In addition, the selection of a model instead of the other can be done taking into account other conformance criteria, e.g., not choosing a model that decrease the Structural conformance dimension.

finishConditionReached
The loop of the algorithm is executed until some finishing conditions are reached, e.g., the precision of the refined model is perfect, or all the possible points to add a new place has been explored. There are also other conditions based on the user criteria, such as limiting the number of new places added to the original model, or the maximum number of tokens needed for any new place (in case of using non pure firing causalities).

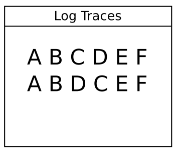
postProcessing
Finally, when the algorithm has finish with the refinement process, the resulting model can be post-processed, e.g., to remove the possible redundant places (i.e., removing do not modify the behavior of the net) appeared after adding new places [38].

In the remaining of this section we present an example (cf. Fig. 7.6) to illustrate the approach presented. Given the log in Fig. 7.6(a) and the model in Fig. 7.6(b), we perform the precision conformance analysis and we realize that the model does not describe precisely the process, i.e., it has some precision issues reflecting the existence of 2 Minimal Disconformant Traces (cf. Fig. (c)). Therefore, we decide to apply the Breaking Concurrencies algorithm in order to improve this precision. First, the log relations are computed (cf. Fig. 7.3). Then, the model concurrencies are computed (cf. Fig. 7.2). Then, we decide to break the concurrency between B and C (a concurrency of the model not reflected in the log) adding a new place, and trying to tackle the Minimal Disconformant Traces AC. We check the conformance of the new model (cf. (d)), and we realize that the precision has improved, but it is not completely precise yet (now there is only one MDT: "ABCE"). Therefore, we decide to continue refining this new model. Among the concurrencies of the new model, we select the one between D and E (not present in the log). Introducing a new place and checking the conformance of the new model (cf. (e)) we see that the model is now completely precise (there are no MDTs), finishing with the refinement process. The last step is to remove the redundant places that have appeared (cf. (f)).

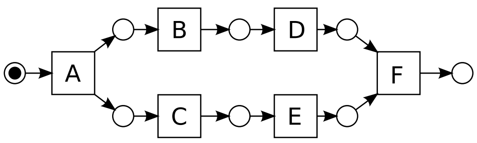
7.2 Supervisory Control Refinement

There are situations where the refinement of the model to increase its precision can not be done simply adding a new place between tasks. These cases need a more complex refinement process, that may involve several tasks and several possible scenarios. Figure 7.7 shows a simple example of one of these possible situations.

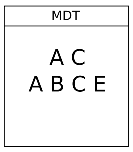
The model shown in the figure has some precision issues with respect to the log, as it is reflected in the set of minimal disconformant traces of Fig. (c). The model allows the execution


Log Traces
A B C D E F
A B D C E F

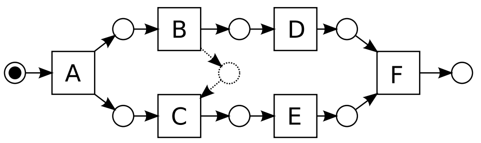
(a) Log



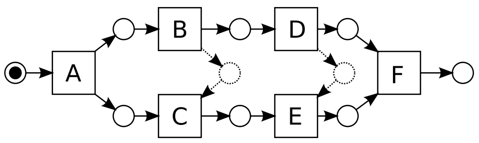
(b) Model



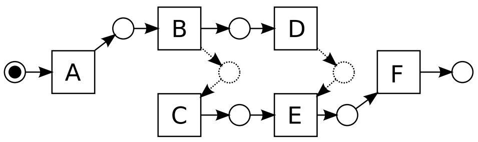
(c) MDT



(d) Refined Model 1

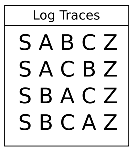


(e) Refined Model 2

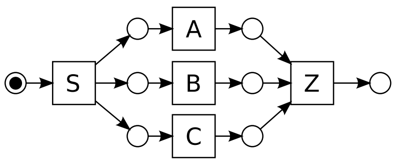


(f) Final Model

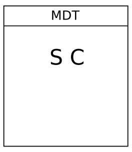
Figure 7.6: *Complete Example of the Breaking Concurrencies Approach. The example is composed by a log (a), a model (b), and the Minimal Disconformant Traces (c) resulting from checking the precision between them. First, the model is refined to break the concurrency btween B and C (d). Then, the model is refined again to break the concurrency between D and E (e). The final model, after removing the redundant places is shown in (f).*


Log Traces
S A B C Z
S A C B Z
S B A C Z
S B C A Z

(a) Log



(b) Model



(c) MDT

Figure 7.7: *Example of imprecision motivating the use of Supervisory Control Refinement.*

of C before firing A or B, but this is a behavior not reflected in the log. However, the *breaking concurrencies* technique presented in the previous section can not be applied in this case, i.e., the addition of a place between tasks A and C would produce a decrease on fitness, not being able to reproduce the trace SBCAZ any more (similar for the case between B and C). For that reason, in this section we propose the use of the the *Supervisory Control Theory* to create a more complex refinement element, called *supervisor*, in order to restrict the model, increasing the precision, and being then a more accurate reflex of the log.

In *Supervisory Control* [20], a *supervisor* is a Discrete-event system (DES)[2] that controls an other DES (the model represented as a Petri net in our case), called the *plant*. The composition of supervisor and plant is called *the closed loop*. The goal of the supervisor is to ensure that some plant requirements are satisfied. In our case, these requirements concern about the precision, and the avoidance of precision discrepancies between the model and the log. A supervisor restricts the Petri net plant by restricting the set of enabled transitions. This is done

[2]Systems with dynamics driven by the occurrence of events, such as Petri nets and automatons.

dynamically, based on the execution of events observed in the plant. Fig. 7.8 shows a diagram of the Supervisory Control Architecture.

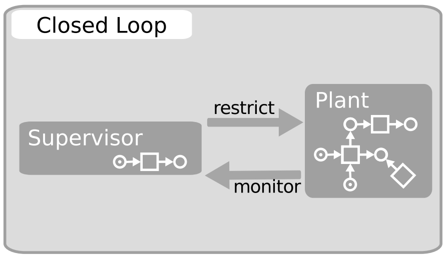
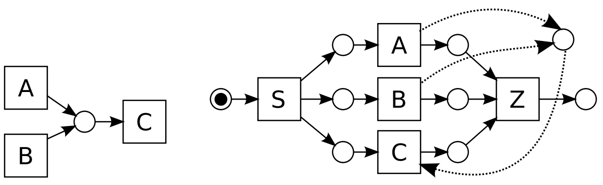


Figure 7.8: *Architecture of Supervisory Control*

Following with the previous example, we use the Supervisory Control framework to derive a supervisor for controlling the precision, without losing fitness. In this case, the supervisor will be a Petri net too. Fig. 7.9(a). shows a possible supervisor, designed to control the precision problem pointed by the only minimal disconformant trace (SC). Fig. 7.9(b) shows the closed loop, once the plant and the supervisor has been composed. The conformance analysis applied to the new refined model determines that both precision and fitness are perfect, i.e., the model describes precisely the log without loosing fitness.



(a) Supervisor (b) Closed Loop

Figure 7.9: *Example of Supervisory Control Refinement. Fig. (a) shows a possible supervisor, and Fig. (b) shows the closed loop, once the supervisor and the plant has been composed.*

The example show above is a simple example with only one minimal disconformant trace, used to illustrate and motivate the application of Supervisory Control strategies. However, in real case scenarios, the number of minimal disconformant traces could be huge, being difficult to use them directly to generate the supervisors. For that reason, could be necessary some pre-processing of the MDTs, abstracting and deriving some rules pointing out the precision issue we want to solve (e.g., the extra behavior generated by an specific loop), and then generating a supervisory to *cover* these rules. Fig. 7.10 illustrates this scenario. An example of this kind of situations could be the extra behavior allowed by a loop in the model, and not reflected in the log. This extra behavior of the loop would be reflected in a large number of MDTs. Abstracting from the MDT we can derive a rule focus on the loop, and cover this rule with a supervisor that controls the loop.

The Supervisory Control Theory [20] is a wide area, where a lot has been accomplished already. Different techniques has been used for deriving supervisors, e.g., the use of the Theory of Regions for generate Petri net supervisors, presented in [40]. Supervisory Control has been applied for a lot of purposes, e.g., the prevention of deadlock for flexible manufacturing systems [40]. It has also been applied in the Process Mining area, e.g., in [36] the authors propose a supervisory control service architecture to support end users of flexible Process-Aware Information Systems during process execution. Therefore, the application of Supervisory Control for

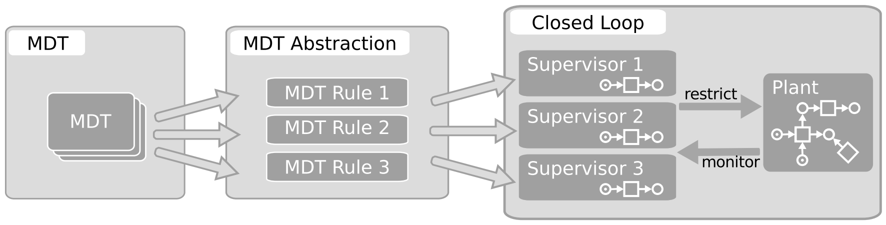


Figure 7.10: *Scenario for the use of MDT in the Supervisory Control Refinement process.*

Process Refinement requires a whole complete research study, and consequently, this section should be seen as the first stone of the process, pointing out the direction for future research work efforts.

8 Case Study

This chapter presents a complete case study, i.e., a complete and real-like example illustrating the application of the procedures and approaches presented in this thesis.

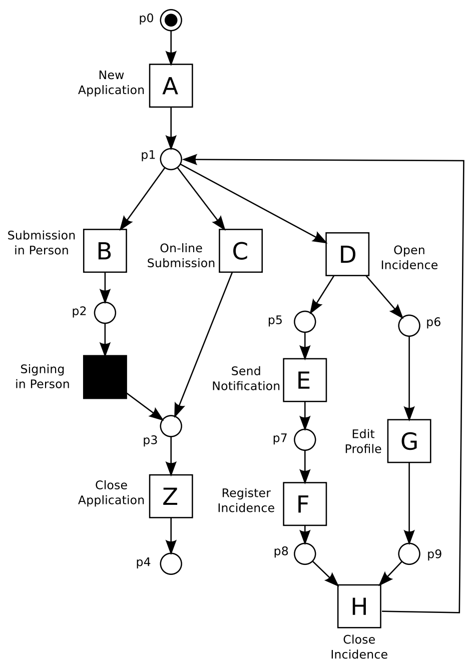
In this case study we analyze the process called *Application*. This process correspond to the process of handing in an application form in a university context. The application procedure can be done on-line or in person. In addition, there are incidences than need to be solve before continue with the application procedure.

The university uses a software product to manage the application process. In order to control processes, this software requires a control-flow model (a Petri net in this case) of each process describing the possible paths it can take. This flow information is used to enable or disable possible actions in the software according to the actual situation of the application process, reducing errors and time, and facilitating the work of the university staff.

Our objective, shown in this case study, is to check the precision of the model used by the software with respect to the application process. In case of not been precise enough, we are interested in refining the model in order to obtain a more accurate model.

In order to perform the conformance checking, first we need a realistic reflex of the process. As in all Process Mining techniques, we use the Event Logs as an unbiased and accurate representation of the process. In this case, the software used by the university to control the application process registers any event or action happened in the system related with the process. The list of events that can be appear in the logs is the followings: *New Application*, *Submission in Person*, *On-lin Submission*, *Open Incidence*, *Send Notification*, *Edit Profile*, *Register Incidence*, *Close Incidence*, *View Profile*, and *Close Application*.

The next step in order to perform a conformance analysis is to map the events in the log and the transitions in the Petri net model. The mapping in this case is is quite straightforward (cf. Fig. 8.1). There is only two remarkable situations. The first one is that the action *View Profile* is a *Non Modeled Task*, i.e, it represents an action that can be done in any moment without affecting any aspect of the process, and therefore it does not correspond with a step of the application procedure. For that reason the log is pre-processed, removing all the occurrences of that event. The second remarkable situation concerns about the application done in person. When a student choose this option, the application must be signed in person, in front of the university staff. This step is not detected by the system, because is done manually. However, given that the signing is compulsory, the model must include this step. This is done defining the action as an *Invisible Task*. The model, the log, and the mapping between them can be seen in Fig. 8.1.



(a) Model

# Instances	Log Traces
183	A C Z
249	A B Z
330	A D E G F H B Z
170	A D E G F H C Z

(b) Log

Figure 8.1: *Model and Log of the Case Study, and the Mapping between them.*

Once the mapping between model and log is done, the next step is to begin the conformance analysis. The first thing to be check is the fitness. In this case, there is complete fitness between log and model, i.e., the model is able to reproduce any trace in the log. This result is expected given that the software uses the model in order to enable or disable possible actions, and consequently, the order of the events in the log is strong tied with respect to the model.

After ensuring the fitness of the system, we continue with the precision conformance analysis. This step is perform using the approach presented in Sec. 4. Figure 8.2(a) shows the details of the log-based traversal of the model behavior.

After performing the conformance analysis, we obtain two results. The first one is the value of the *ETC Precision metric*. In this case, the value returned is 0.83, indicating a high precision level of the model, but not completely. The second result is the set of *Minimal Disconformant Traces* locating where are the precision problems. In this case, there are three MDTs (cf. Fig. 8.2(b)) pointing directly to the three Escaping Edges in the system. Analyzing the set of MDT we realize that all the precision issues are produced by the management of incidences in the application procedure. In particular:

- The Minimal Disconformant Traces "ADEGFHD" refers to the possibility of detecting several incidences in the same application. This is a behavior allowed by the model but not reflected in the log. However, after consulting with an expert in the application domain, we decide that this is a possibility that must exist: the absence of several incidences can be result of a excellent work of the university staff, but there is no guarantee that this situation does not occur in the future.

- The other two MDT refer to the part of the application process where the incidence is managed. It seems this part is not accurate with the reality, and we decide to refine the

State ID	State Prefix	Allowed Tasks (AT)	Reflected Tasks (RT)	Escaping Edges (EE)
1		A	A	
2	A	B C D	B C D	
3	AC	Z	Z	
4	ACZ			
5	AB	Z	Z	
6	ABZ			
7	AD	E G	E	G
8	ADE	F G	G	F
9	ADEG	F	F	
10	ADEGF	H	H	
11	ADEGFH	B C D	B C	D
12	ADEGFHB	Z	Z	
13	ADEGFHBZ			
14	ADEGFHC	Z	Z	
15	ADEGFHCZ			

(a) Traversal Details

MDT
A D G
A D E F
A D E G F H D

(b) MDT

Figure 8.2: *Details of the Log-Based Traversal of the Case Study Model Behavior.This Figure include also the* MDT *detected during the precision analysis.*

model to improve the precision.

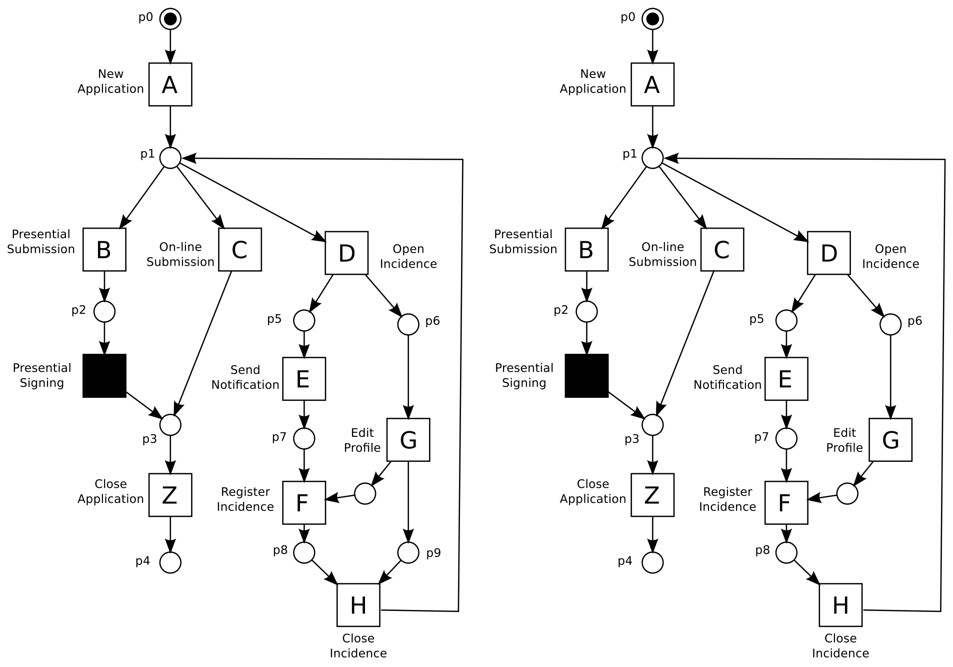
The approach used to refine the precision of the model is the *Breaking Concurrencies* technique, seen in Section 7.1. The first step of this technique is to compute the log and the model relations in order to decide the possible concurrencies to break. Fig. 8.3 shows the Firing Causality Matrix for the log perspective. For the model perspective, we detect the concurrencies E-G and F-G.

	A	B	C	D	E	F	G	H	Z
A	-	1	1	1	1	1	1	1	1
B	0	-	1	1	1	1	1	1	1
C	0	1	-	1	1	1	1	1	1
D	0	1	1	-	1	1	1	1	1
E	0	1	1	0	-	1	1	1	1
F	0	1	1	0	0	-	0	1	1
G	0	1	1	0	0	1	-	1	1
H	0	1	1	0	0	0	0	-	1
Z	0	0	0	0	0	0	0	0	-

Figure 8.3: *Firing Causality Matrix for the log in Case Study.*

After analyzing the possible concurrencies to be broken, we decide not to tackle the F-G concurrency. This concurrent behavior is not reflected in the log. However, this is not produced for not being concurrent steps in the process, but for the time needed to perform each task. *Edit Profile* is a time-consuming task that require the participation of a university staff person to correct the profile. On the other hand, the task *Send Notification* is an instantaneous action performed in background by the own software. Therefore, it is compressible not to find concurrency between both tasks, but they should be done concurrently because there is no relation between the notification and the profile editing.

Hence, we decide to tackle the F-G concurrency, allowed by the model but not reflected in the log. To do that, we introduce a new place from G to F, as it is shown in Fig. 8.4(a). After checking the conformance of the new model, we realize that the precision has improved (the etc$_P$ value of the new model is 0.89), and therefore there are less MDT (the trace ADEF is no longer in the set). Thus, we decide to choose the new model instead of the old one.



(a) Refined Model (b) Final Model

Figure 8.4: *Refined and Final Model of the Case Study. Figure (a) shows the refined model, after breaking the concurrency between F and G. Figure (b) shows the post-processed version of the refined model after removing the redundant place between G and H.*

Finally, after finishing the refinement process, we decide to post-process the model in order to remove the redundant place between G and H. The final refined model can be seen in Fig. 8.4(b).

This new model describes more precisely than the old one the process Application. Moreover, the precison issues has been analyzed during the precision conformance procedure, and all them correspond to accepted generalizations of the model.

9 Conclusions and Future Work

This work presents an approach to analysis the conformance between a model and a log. In particular, the approach presented is addressed to check the *precision* of a Petri net with respect to a log. The technique is based on the effort needed to obtain a completely precise model as an estimator for precision. By only focusing on the underlying behavior of the Petri net that is reflected in the log, the technique avoids the potential state explosion that might arise when dealing with large and highly concurrent nets.

The technique provides a metric, called *ETC Precision* (etc_P), to quantify the degree of precision. The technique is also enriched with the computation of the *Minimal Disconformant Traces* (MDT), a set of traces used to identify the exact points where the model behavior begins to deviate from de log behaviors, denoting a precision problem. The Minimal Disconformant Traces may be also the starting point for refine the model to better represent the log.

The precision analysis approach has been implemented as a plug-in within ProM 6, a open-source framework for implementing process mining tools in a standard environment. This implementation has been used to perform precision analysis in a wide set of benchmarks, obtaining promising experimental results, especially for large benchmarks (intractable by the current approaches).

Together with the conformance approach, this thesis has included approaches for the refinement of models, in order to achieve a more accurate model of the reality. These techniques are especially focus on fixing the precision issues of the model, using the precision analysis results obtained by the conformance approach presented in this thesis.

One technique presented for refining precision, called *Breaking Concurrencies*, addresses the problem of the precision issues produced by concurrencies in the model allowing a behavior not reflected in the log. The approach presented tries to locate these concurrencies and *break* them, introducing a new constraint to restrict this extra behavior, improving the precision.

Finally, this thesis has proposed the use of the *Supervisory Control Theory* as a mechanism to correct and refine conformance problems. In particular, this work propose the use of the Minimal Disconformance Traces obtained in the precision analysis, in order to refine the precision of the model. The application of Supervisory Control seems to be a promising area for process refinement, addressing the problem in a more global way, and being able to detect and correct problems than other approach cannot. For that reason, the future efforts should be focus on this direction, studying and elaborating techniques to build supervisors for controlling the precision of a model, but also other conformance dimensions.

Bibliography

[1] Extensible Markup Language (XML). http://www.w3.org/XML.

[2] MXML Schema. http://prom.win.tue.nl/tools/promimport.

[3] Process Mining. http://www.processmining.org.

[4] ProM Framework. http://prom.win.tue.nl/tools/prom.

[5] ProM Import. http://prom.win.tue.nl/tools/promimport.

[6] Calders, T., Günther, C. W., Pechenizkiy, M., and Rozinat, A. Using minimum description length for process mining. In *SAC* (2009), S. Y. Shin and S. Ossowski, Eds., ACM, pp. 1451–1455.

[7] Carmona, J., Cortadella, J., and Kishinevsky, M. Genet: a tool for the synthesis and mining of petri nets (tool paper). In *Proceedings of the 2009 Application of Concurrency to System Design conference (ACSD 2009)* (Augsburg, Germany, July 2009), pp. 181–185.

[8] Cook, J. E., and Wolf, A. L. Event-based detection of concurrency. In *SIGSOFT FSE* (1998), pp. 35–45.

[9] Cook, J. E., and Wolf, A. L. Software process validation: Quantitatively measuring the correspondence of a process to a model. *ACM Trans. Softw. Eng. Methodol. 8*, 2 (1999), 147–176.

[10] Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., and Yakovlev, A. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems 3*, E80-D (1997), 315–325.

[11] de Medeiros, A. K. A. *Genetic Process Mining.* PhD thesis, Technische Universiteit Eindhoven, 2006.

[12] de Medeiros, A. K. A., van der Aalst, W. M. P., and Weijters, A. J. M. M. Quantifying process equivalence based on observed behavior. *Data Knowl. Eng. 64*, 1 (2008), 55–74.

[13] de Medeiros, A. K. A., van Dongen, B. F., van der Aalst, W. M. P., and Weijters, A. J. M. M. Process mining for ubiquitous mobile systems: An overview and a concrete algorithm. In *UMICS* (2004), L. Baresi, S. Dustdar, H. Gall, and M. Matera, Eds., vol. 3272 of *Lecture Notes in Computer Science*, Springer, pp. 151–165.

[14] de Medeiros, A. K. A., Weijters, A. J. M. M., and van der Aalst, W. M. P. Genetic process mining: an experimental evaluation. *Data Min. Knowl. Discov. 14*, 2 (2007), 245–304.

[15] Dumas, M., van der Aalst, W. M. P., and ter Hofstede, A. H. M. *Process-Aware Information Systems: Bridging People and Software Through Process Technology.* Wiley-Blackwell, Oct. 2005.

[16] Esparza, J. A polynomial-time algorithm for checking consistency of free-choice signal transition graphs. *Fundam. Inform. 62*, 2 (2004), 197–220.

[17] Folino, F., Greco, G., Guzzo, A., and Pontieri, L. Discovering expressive process models from noised log data. In *IDEAS* (2009), B. C. Desai, D. Saccà, and S. Greco, Eds., ACM International Conference Proceeding Series, ACM, pp. 162–172.

[18] Greco, G., Guzzo, A., Pontieri, L., and Saccà, D. Discovering expressive process models by clustering log traces. *IEEE Trans. Knowl. Data Eng. 18*, 8 (2006), 1010–1027.

[19] Günther, C. W., and van der Aalst, W. M. P. Fuzzy mining - adaptive process simplification based on multi-perspective metrics. In *BPM* (2007), G. Alonso, P. Dadam, and M. Rosemann, Eds., vol. 4714 of *Lecture Notes in Computer Science*, Springer, pp. 328–343.

[20] Iordache, M. V., and Antsaklis, P. J. *Supervisory Control of Concurrent Systems: A Petri Net Structural Approach.* Birkhäuser, 2006.

[21] Kovalyov, A. Concurrency relations and the safety problem for petri nets. In *Application and Theory of Petri Nets* (1992), K. Jensen, Ed., vol. 616 of *Lecture Notes in Computer Science*, Springer, pp. 299–309.

[22] Kovalyov, A., and Esparza, J. A polynomial algorithm to compute the concurrency relation of free-choice signal transition graphs. In *Proc. Intl. Workshop on Discrete Event Systems* (1996), Citeseer, pp. 1–6.

[23] Li, J., Liu, D., and Yang, B. Process mining: Extending alpha-algorithm to mine duplicate tasks in process logs. In *APWeb/WAIM Workshops* (2007), K. C.-C. Chang, W. Wang, L. C. 0002, C. A. Ellis, C.-H. Hsu, A. C. Tsoi, and H. Wang, Eds., vol. 4537 of *Lecture Notes in Computer Science*, Springer, pp. 396–407.

[24] Mans, R. S., Schonenberg, H., Song, M., van der Aalst, W. M. P., and Bakker, P. J. M. Application of process mining in healthcare - a case study in a dutch hospital. In *BIOSTEC (Selected Papers)* (2008), A. L. N. Fred, J. Filipe, and H. Gamboa, Eds., vol. 25 of *Communications in Computer and Information Science*, Springer, pp. 425–438.

[25] Muñoz-Gama, J., and Carmona, J. A fresh look at precision in process conformance. In *Business Process Management* (Sept. 2010), Lecture Notes in Computer Science, Springer.

[26] Murata, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE 77*, 4 (April 1989), 541–580.

[27] Peterson, J. L. Petri nets. *ACM Computing Surveys 9*, 3 (September 1977), 223–252.

[28] Pinter, S. S., and Golani, M. Discovering workflow models from activities' lifespans. *Computers in Industry 53*, 3 (2004), 283–296.

[29] Rozinat, A., de Medeiros, A. K. A., Günther, C. W., Weijters, A. J. M. M., and van der Aalst, W. M. P. The need for a process mining evaluation framework in research and practice. In *Business Process Management Workshops* (2007), A. H. M. ter Hofstede, B. Benatallah, and H.-Y. Paik, Eds., vol. 4928 of *Lecture Notes in Computer Science*, Springer, pp. 84–89.

[30] Rozinat, A., de Medeiros, A. K. A., Günther, C. W., Weijters, A. J. M. M., and van der Aalst, W. M. P. Towards an evaluation framework for process mining algorithms. *BPM Center Report BPM-07-06, BPMcenter. org* (2007).

[31] Rozinat, A., and van der Aalst, W. M. P. Conformance testing: measuring the alignment between event logs and process models. *BETA Working Paper Series, Eindhoven University of Technology WP 144* (2005), 203–210.

[32] Rozinat, A., and van der Aalst, W. M. P. Decision mining in prom. In *Business Process Management* (2006), S. Dustdar, J. L. Fiadeiro, and A. P. Sheth, Eds., vol. 4102 of *Lecture Notes in Computer Science*, Springer, pp. 420–425.

[33] Rozinat, A., and van der Aalst, W. M. P. Conformance checking of processes based on monitoring real behavior. *Information Systems 33*, 1 (2008), 64–95.

[34] Rozinat, A., Veloso, M., and van der Aalst, W. M. P. Evaluating the quality of discovered process models. In *Proceedings of Induction of Process Models (IPM workshop at ECML PKDD 2008, September 15, Antwerp, Belgium)* (2008), vol. 16, pp. 45–52.

[35] Rozinat, A., Veloso, M., and van der Aalst, W. M. P. Using hidden markov models to evaluate the quality of discovered process models. *Extended Version. BPM Center Report BPM-08-10, BPMcenter. org* (2008).

[36] Santos, E. A. P., Francisco, R., Pesic, M., and van der Aalst, W. M. P. Supervisory control service - a control approach supporting flexible processes. Tech. rep., BPM Center Report, BPMcenter.org, Apr. 2010.

[37] Scheer, A.-W. *Aris–Business Process Modeling.* Springer-Verlag, 1998.

[38] Silva, M., Teruel, E., and Colom, J. M. Linear algebraic and linear programming techniques for the analysis of place or transition net systems. In *Petri Nets* (1996), W. Reisig and G. Rozenberg, Eds., vol. 1491 of *Lecture Notes in Computer Science*, Springer, pp. 309–373.

[39] Solé, M., and Carmona, J. Process mining from a basis of state regions. In *Petri Nets* (2010), J. Lilius and W. Penczek, Eds., vol. 6128 of *Lecture Notes in Computer Science*, Springer, pp. 226–245.

[40] Uzam, M. An optimal deadlock prevention policy for flexible manufacturing systems using petri net models with resources and the theory of regions. *The International Journal of Advanced Manufacturing Technology 19*, 3 (2002), 192–208.

[41] van der Aalst, W. M. P. Formalization and verification of event-driven process chains. *Information & Software Technology 41*, 10 (1999), 639–650.

[42] van der Aalst, W. M. P., de Medeiros, A. K. A., and Weijters, A. J. M. M. Genetic process mining. In *ICATPN* (2005), G. Ciardo and P. Darondeau, Eds., vol. 3536 of *Lecture Notes in Computer Science*, Springer, pp. 48–69.

[43] van der Aalst, W. M. P., Reijers, H. A., and Song, M. Discovering social networks from event logs. *Computer Supported Cooperative Work 14*, 6 (2005), 549–593.

[44] van der Aalst, W. M. P., Rubin, V., Verbeek, H. M. W. E., van Dongen, B. F., Kindler, E., and Günther, C. W. Process mining: a two-step approach to balance between underfitting and overfitting. *Software and Systems Modeling* (2009).

[45] van der Aalst, W. M. P., van Dongen, B. F., Herbst, J., Maruster, L., Schimm, G., and Weijters, A. J. M. M. Workflow mining: A survey of issues and approaches. *Data Knowl. Eng. 47*, 2 (2003), 237–267.

[46] van der Aalst, W. M. P., Weijters, A. J. M. M., and Maruster, L. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering 16*, 9 (September 2004), 1128–1142.

[47] van der Werf, J. M. E. M., van Dongen, B. F., Hurkens, C. A. J., and Serebrenik, A. Process discovery using integer linear programming. In *Petri Nets* (2008), K. M. van Hee and R. Valk, Eds., vol. 5062 of *Lecture Notes in Computer Science*, Springer, pp. 368–387.

[48] van Dongen, B. F., de Medeiros, A. K. A., and Wen, L. Process mining: Overview and outlook of petri net discovery algorithms. *T. Petri Nets and Other Models of Concurrency 2* (2009), 225–242.

[49] van Dongen, B. F., Mendling, J., and van der Aalst, W. M. P. Structural patterns for soundness of business process models. In *EDOC* (2006), IEEE Computer Society, pp. 116–128.

[50] van Dongen, B. F., and van der Aalst, W. M. P. Multi-phase process mining: Building instance graphs. In *ER* (2004), P. Atzeni, W. W. Chu, H. Lu, S. Zhou, and T. W. Ling, Eds., vol. 3288 of *Lecture Notes in Computer Science*, Springer, pp. 362–376.

[51] Weijters, A. J. M. M., and van der Aalst, W. M. P. Rediscovering workflow models from event-based data using little thumb. *Integrated Computer-Aided Engineering 10*, 2 (2003), 151–162.

[52] Weijters, A. J. M. M., van der Aalst, W. M. P., and de Medeiros, A. K. A. Process mining with the heuristicsminer algorithm. *BETA Working Paper Series WP 166* (2006).

[53] Wen, L., van der Aalst, W. M. P., Wang, J., and Sun, J. Mining process models with non-free-choice constructs. *Data Min. Knowl. Discov. 15*, 2 (October 2007), 145–180.

[54] Wen, L., Wang, J., and Sun, J. Mining invisible tasks from event logs. In *AP-Web/WAIM* (2007), G. Dong, X. Lin, W. W. 0011, Y. Yang, and J. X. Yu, Eds., vol. 4505 of *Lecture Notes in Computer Science*, Springer, pp. 358–365.

[55] Wen, L., Wang, J., van der Aalst, W. M. P., Huang, B., and Sun, J. A novel approach for process mining based on event types. *J. Intell. Inf. Syst. 32*, 2 (2009), 163–190.