

Multi-perspective process mining

Citation for published version (APA):

Mannhardt, F. (2018). Multi-perspective process mining Eindhoven: Technische Universiteit Eindhoven

Document status and date:

Published: 07/02/2018

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

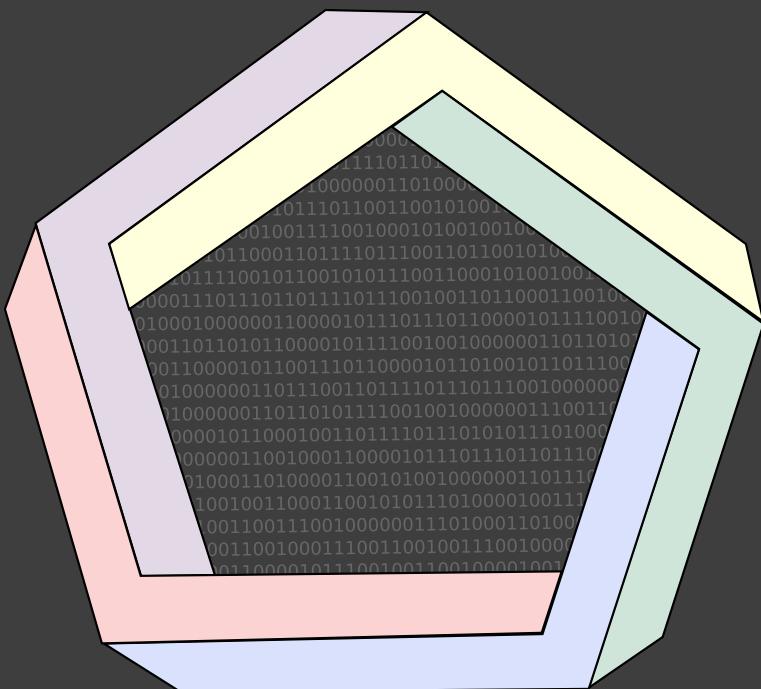
Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Multi-perspective Process Mining



Felix Mannhardt

Multi-perspective Process Mining

Copyright © 2018 by F. Mannhardt. All Rights Reserved.

Mannhardt, F.

Multi-perspective Process Mining Technische Universiteit Eindhoven, 2018.
Proefschrift.

Keywords: Process mining, Conformance checking, Process discovery, Multiple perspectives, Event logs

A catalogue record is available from the Eindhoven University of Technology Library

ISBN 978-90-386-4438-7

SIKS Dissertation Series No. 2018-02



The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Cover released under CC BY-SA. Original image credits: Penrose pentagon by Trilink at English Wikipedia (CC BY-SA):
https://commons.wikimedia.org/wiki/File:Penrose_pentagon.svg

This thesis has been created using LuaL^AT_EX.

Multi-perspective Process Mining

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baajens, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op woensdag 7 februari 2018 om 16:00 uur

door

Felix Mannhardt

geboren te Neuss, Duitsland

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter: prof.dr.ir. J.J. Lukkien
1^e promotor: prof.dr.ir. Hajo A. Reijers
2^e promotor: prof.dr.ir. Wil M.P. van der Aalst
copromotor: dr. Massimiliano de Leoni
leden: prof.dr. Jan Vanthienen (Katholieke Universiteit Leuven)
prof.dr. Pieter J. Toussaint (Norges Teknisk-naturvitenskapelige
Universitet)
dr.ir. Remco M. Dijkman
dr. Josep Carmona Vargas (Universitat Politècnica de Catalunya)

Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

Dedicated to Danni and Joris.

Abstract

This thesis is about process mining: the analysis an organization's processes by using process execution data. During the handling of a case or process instance data about the execution of activities is recorded in databases. We use such process execution data to gain insights about the real execution of processes. In this thesis, we address research challenges in which a multi-perspective view on processes is needed and that look beyond the control-flow perspective, which defines the sequence of activities of a process. We consider problems in which multiple interacting process perspectives — in particular control-flow, data, resources, time, and functions — are considered together. We propose five multi-perspective process mining methods that deal with the interaction of multiple process perspectives:

- A conformance checking method that balances the importance of multiple perspectives to provide an alignment between recorded event data and a process model. The method provides reliable diagnostics and quality measures with respect to all perspectives of the process model
- A precision measure for multi-perspective process models with regard to an event log. The precision of a process model is determined as the fraction of the behavior possible according to the model in relation to what has actually been observed in the event log.
- A process discovery method that uses domain knowledge on the functional perspective of a process to improve the result of existing discovery methods. The domain knowledge is expressed as a set of multi-perspective activity patterns and a mapping between low-level events and instantiations of the activity patterns is computed. By grouping low-level events to instances of recognizable activities on a higher abstraction level the understanding of automatically discovered process models by stakeholders is facilitated.
- A process discovery method that uses the data perspective of a process to distinguish certain infrequent paths from random noise. The method discovers infrequent paths that can be characterized by rules by employing classification techniques. The data perspective is used to improve the discovered control-flow: Data- and control-flow are learned together.
- A decision mining method for discovering, potentially, overlapping decision rules. In contrast to existing methods for the same data values more than one alternative activity may be activated at a decision point.

All methods have been implemented, systematically evaluated, and applied in real-life situations in the context of four case studies.

Contents

Contents	ix
List of Figures	xv
List of Tables	xxiii
List of Algorithms	xxv
I Introduction	1
1 Introduction	3
1.1 Process Mining	5
1.2 One Process – Multiple Perspectives	9
1.2.1 Multi-perspective Process Models and Event Logs	10
1.2.2 Five Considered Perspectives	11
1.2.3 Relation to Existing Multi-perspective Frameworks	13
1.3 Research Goals and Contributions	14
1.3.1 Overall Research Goals	14
1.3.2 Contributions	15
1.4 Structure	17
2 Preliminaries	19
2.1 Basic Notations	19
2.2 Variables and Guard Expressions	22
2.3 Event Logs	24
2.4 Decision Trees	27
3 Process Models	29
3.1 Process Behavior Expressed as a Trace Set	30
3.2 Data Petri Nets	32
3.2.1 Petri Nets	32
3.2.2 Syntax and Semantics of DPNs	34
3.3 Causal Nets	41
3.4 BPMN and Extended Data Petri Nets	46
3.4.1 BPMN	46
3.4.2 Extended Data Petri Nets	47

II Multi-perspective Conformance	49
4 Introduction to Multi-perspective Conformance Checking	51
4.1 Conformance Checking	52
4.2 Multi-perspective Conformance Checking	56
4.3 Aligning Process Models and Event Logs	58
4.3.1 Alignments	58
4.3.2 Optimal Alignment and Cost Function	62
4.3.3 Choice of a Cost Function	63
4.4 Measuring Fitness Based on Alignments	64
5 Multi-perspective Alignment	67
5.1 Motivation for Balanced Alignments	67
5.2 Balanced Alignment Method	68
5.2.1 Assumptions on the Input	69
5.2.2 A* Algorithm and Search Space	70
5.2.3 Searching an Optimal Balanced Alignment	72
5.2.4 Formal Guarantees	78
5.2.5 Computing an Optimal Variable Assignment	82
5.2.6 Computational Complexity	88
5.2.7 Optimizations	91
5.3 Evaluation	95
5.3.1 Datasets and Experimental Setup	97
5.3.2 Results	98
5.4 Related Work	106
5.4.1 Control-flow Conformance Checking	106
5.4.2 Multi-perspective Conformance Checking	107
5.5 Conclusion	109
5.5.1 Contribution	109
5.5.2 Limitations	110
5.5.3 Future Work	111
6 Multi-perspective Precision	113
6.1 Motivation for a Multi-perspective Precision Measure	114
6.2 Multi-perspective Precision Measure	116
6.2.1 Assumptions on the Input	117
6.2.2 Precision Measure	118
6.2.3 Activity-precision Measure	121
6.2.4 Resource-precision Measure	125
6.3 Locating Precision Problems	127
6.4 Evaluation	129
6.5 Related Work	132
6.6 Conclusion	133
6.6.1 Contribution	133

6.6.2	Limitations	134
6.6.3	Future Work	135
III	Multi-perspective Discovery and Enhancement	137
7	Introduction to Multi-perspective Discovery and Enhancement	139
7.1	Process Discovery and Enhancement	139
7.1.1	Process Discovery	140
7.1.2	Process Enhancement	141
7.2	Challenges for Process Discovery Methods	141
7.2.1	Incompleteness	142
7.2.2	Noise	143
7.2.3	Granularity	144
7.3	Multi-perspective Process Discovery	144
8	Data-aware Heuristic Process Discovery	147
8.1	Motivation for Discovering Conditional Behavior	148
8.2	Overview of the Data-aware Heuristic Process Discovery Method .	150
8.3	Discovering Conditional Behavior	151
8.3.1	Dependency Conditions and Dependency Measure	151
8.3.2	Discovering Dependency Conditions	154
8.3.3	Discovering Causal Nets with the DHM	157
8.4	Extending Causal Nets with Multiple Perspectives	163
8.4.1	Data Causal Nets	163
8.4.2	Discovering DC-Nets	168
8.5	Evaluation	171
8.5.1	Event Log and Methods	171
8.5.2	Experimental Design	171
8.5.3	Results	172
8.6	Related Work	174
8.6.1	Noise Filtering Techniques	174
8.6.2	Multi-perspective Discovery	175
8.7	Conclusion	176
8.7.1	Contribution	176
8.7.2	Limitations	176
8.7.3	Future Work	177
9	Guided Multi-perspective Process Discovery	179
9.1	Motivation for Event Abstraction	180
9.2	Guided Process Discovery Method	183
9.2.1	Overview of the GPD Method	183
9.2.2	Encoding the High-level Behavior in Activity Patterns	184
9.2.3	Identifying the Activity Patterns	187
9.2.4	Composing the Activity Patterns to an Abstraction Model .	189

9.2.5 Aligning the Event Log and the Abstraction Model	195
9.2.6 Abstracting the Event Log	197
9.2.7 Discovering a High-Level Process Model	200
9.2.8 Expanding the High-level Activities and Validating the Model	201
9.3 Implementation	203
9.4 Evaluation	204
9.5 Related Work	206
9.6 Conclusion	208
9.6.1 Contribution	208
9.6.2 Limitations	209
9.6.3 Future Work	209
10 Enhancing Models with Overlapping Decision Rules	211
10.1 Introduction to Decision Mining	211
10.1.1 Decision Rules	211
10.1.2 Mining Decision Rules	213
10.2 Motivation for Overlapping Decision Rules	215
10.3 Discovery of Overlapping Decision Rules	217
10.3.1 Parameters and Assumptions on the Input	217
10.3.2 Overall Decision Procedure	218
10.3.3 Building Overlapping Guard Expressions	221
10.3.4 Dealing With Real-life Event Logs	225
10.4 Evaluation	225
10.4.1 Evaluation Setup	226
10.4.2 Results and Discussion	228
10.5 Related Work	231
10.6 Conclusion	233
10.6.1 Contribution	233
10.6.2 Limitations	234
10.6.3 Future Work	234
IV Applications	237
11 Tool Support	239
11.1 Interactive Data-aware Heuristic Miner	239
11.1.1 Overview of the iDHM	240
11.1.2 Walk-through of the iDHM	240
11.1.3 Plug-in Architecture	244
11.1.4 Conclusion of the iDHM	245
11.2 Multi-perspective Process Explorer	245
11.2.1 Overview of the MPE	246
11.2.2 Walk-through of the MPE	248
11.2.3 Conclusion of the MPE	258

11.3 Conclusion	259
12 Case Study: Road Traffic Fine Management	261
12.1 Case Description	261
12.1.1 Process Questions	262
12.1.2 Event Log	262
12.1.3 Normative Process Model	264
12.2 Conformance Checking	265
12.2.1 Configuration Settings and Cost Function	265
12.2.2 Conformance Checking Results	266
12.2.3 Comparison With the Non-Balanced Method	267
12.3 Discovery of the Data Perspective	274
12.4 Data-aware Process Discovery	276
12.4.1 Configuration Settings	276
12.4.2 Discovery Results	276
12.4.3 Comparison with State-of-the-art Techniques	280
12.5 Guided Process Discovery	284
12.5.1 Activity Patterns	284
12.5.2 Discovery Results	285
12.5.3 Comparison with State-of-the-art Methods	286
12.6 Conclusion	287
13 Case Study: Sepsis	289
13.1 Case Description	289
13.1.1 Process Questions	290
13.1.2 Event Log	292
13.1.3 Normative Process Model	294
13.2 Conformance Checking	296
13.2.1 Configuration Settings and Cost Function	296
13.2.2 Conformance Checking Results	296
13.3 Discovery of the Data Perspective	298
13.4 Data-aware Process Discovery	300
13.4.1 Configuration Settings	300
13.4.2 Discovery Results	301
13.5 Guided Process Discovery	302
13.5.1 Activity Patterns	303
13.5.2 Discovery Results	305
13.5.3 Comparison to State-of-the-Art Methods	308
13.6 Conclusion	312
14 Case Study: Digital Whiteboard	313
14.1 Case Description	313
14.1.1 Process Questions	314
14.1.2 Event Log	314

14.2 Abstraction and Guided Process Discovery	314
14.2.1 Abstraction Model	315
14.2.2 Event Abstraction	316
14.2.3 Discovery of the Inductive Miner	319
14.2.4 Conformance Checking of the Discovered Model	322
14.2.5 Dotted Chart Analysis of the High-level Log	324
14.3 Conclusion	324
15 Case Study: Hospital Billing	325
15.1 Case Description	325
15.1.1 Process Questions	326
15.1.2 Event Log	326
15.1.3 Normative Model	326
15.2 Discovery of the Data Perspective	329
15.3 Conformance Checking	331
15.4 Data-aware Process Discovery	334
15.4.1 Configuration Settings	334
15.4.2 Discovery Results	334
15.4.3 Comparison with State-of-the-art Techniques	338
15.5 Conclusion	340
V Closure	341
16 Conclusion	343
16.1 Contributions	343
16.1.1 Multi-perspective Conformance (Part II)	343
16.1.2 Multi-perspective Discovery and Enhancement (Part III) .	345
16.1.3 Applications (Part IV)	346
16.2 Limitations	347
16.3 Future Work	349
16.4 Reflection on the Broader Context	351
Bibliography	353
Index	377
Summary	381
Curriculum vitae	383
Acknowledgments	387
SIKS dissertations	389

List of Figures

1.1	Process execution data is recorded for the executions of activities of a process instance. Process mining leverages such data recorded in event logs to analyze the real execution of the process.	4
1.2	Execution traces recorded by two process instances of a hospital process.	5
1.3	The control-flow of an example hospital process modeled using BPMN.	6
1.4	Overview of the three main categories of process mining [Aal16]. .	7
1.5	Process discovery uses the information stored in the event log to discover a suitable process model. Conformance checking diagnoses deviations between process models and the information stored in the event log.	8
1.6	Multiple perspectives on a process can be used for process mining.	12
1.7	Structure of this thesis in context of the three types of process mining.	18
2.1	A simple event log obtained by transforming the four traces of the example event log shown in table 2.1.	27
3.1	Process model notations used in this thesis.	29
3.2	A Petri net modeling the control-flow perspective of the hospital process.	33
3.3	Multiple perspectives of the hospital process modeled with a DPN.	37
3.4	A labeled DPN describing the hospital process.	41
3.5	The hospital process modeled as C-Net.	42
3.6	Possible routing constructs using a set of bindings in a C-net [Aal16].	43
3.7	A BPMN model of the hospital process.	47
3.8	An eDPN model of the hospital process.	48
4.1	Conformance checking of processes in the context of process mining.	52
4.2	Global conformance checking methods establish a global connection between the events and elements of the process model. . . .	53
4.3	Venn diagram depicting the relation between behavior of the process model, event log, and process system.	55
4.4	Model-based multi-perspective conformance checking needs to consider additional constraints based on data, resources, and time. . .	56

4.5	An alignment maps events to activities of a process model.	58
4.6	Venn diagram illustrating the fitness measure.	64
5.1	Motivation for a multi-perspective, balanced alignment	67
5.2	Overview of the proposed balanced alignment method.	69
5.3	Collapsing the search space by considering only the control-flow successors of a node.	73
5.4	Augmentation with variable assignments of a control-flow successor. .	74
5.5	Search space explored for an optimal alignment of σ_4 and the hospital process.	76
5.6	Simulation of inhibitor arcs using variables and guards of a DPN .	90
5.7	Example credit card application process used to evaluate the balanced alignment.	96
5.8	Computation time of the naïve, optimized, and non-balanced alignment of the credit process, using varying trace length and noise types. The staged approach is unable to compute a solution for some traces.	99
5.9	Computation time of the naïve, optimized, and non-balanced alignment of the hospital process, using varying trace length and noise types.	100
5.10	Computation speedup for different types of noise and varying levels of noise.	102
5.11	Absolute difference between the fitness determined by the balanced alignment method and the non-balanced method.	103
5.12	Fitness level of traces determined by the balanced alignment method for which the non-balanced method [LA13a] could not compute an alignment.	105
6.1	Venn diagram illustrating the precision measure.	113
6.2	Inprecise DPN model N_1 of the credit application process.	115
6.3	Precise DPN model N_2 of the credit application process that uses guards.	116
6.4	Overview of the proposed multi-perspective precision measure. .	117
6.5	Very imprecise DPN model N_3 of the credit application process. .	123
6.6	Perfectly precise DPN model N_4 of the credit application process. .	124
6.7	Local precision measure projected on the places of the imprecise DPN model N_1	128
6.8	Local precision measure projected on the places of the more precise model N_2	129
6.9	Model A discovered by the Inductive Miner on the road-traffic fine event log.	130
6.10	Model B based on the model A with data rules discovered by using the ProM decision miner.	131

7.1	Discovery and Enhancement in the context of process mining [Aal16].	139
7.2	Goal of process discovery: discover the, unknown, original process system.	142
7.3	Incompleteness, noise, and the granularity of events are challenges for process discovery methods when working with real-life event logs.	143
7.4	Multi-perspective process discovery methods use information about the context of process to obtain a more complete picture of the process.	145
8.1	Infrequent process behavior highlighted on a BPMN model of the hospital process.	148
8.2	Models discovered by Inductive Miner and Heuristics Miner on an event log with noise.	149
8.3	Overview of the proposed data-aware heuristic process discovery method.	150
8.4	The simplified variant of the hospital process modeled as C-Net.	158
8.5	Illustration of the dependency relation discovery in three steps.	160
8.6	Multiple perspectives of the simplified hospital process modeled as a DC-net.	164
8.7	Conversion from the DC-Net of the hospital process to a DPN that over-approximates its behavior.	169
8.8	Graph edit distance (GED) between the relations discovered by the compared methods and relations of the reference model.	173
9.1	Mapping between events recording the occurrence of low-level activities and instances of the actual high-level activities that were executed for a process instance.	180
9.2	Overview of the seven steps of the proposed GPD method.	184
9.3	Three activity patterns $ap_a, ap_b, ap_c \in AP$ for the example with process models in DPN notation.	186
9.4	An activity pattern capturing the life-cycle of the high-level activity X-Ray.	188
9.5	Overview of the graphical notation for the composition functions.	192
9.6	Abstraction model cp created by composing the patterns ap_a, ap_b , and ap_c .	192
9.7	Implementation of the composition functions using the DPN notation	193
9.8	DPN created by our implementation for the abstraction model cp .	195
9.9	A discovered high-level model in DPN notation.	201
9.10	Expansion of a single high-level activity a with input places i_1, \dots, i_n and output places o_1, \dots, o_n in the discovered model with the DPN modeling the activity pattern ap_a .	202

9.11	A partially expanded high-level model. Activity Shift in the high-level model has been replaced by the DPN that models activity pattern ap_a	203
9.12	A RapidMiner workflow that implements the GPD method.	204
9.13	Average computation time per trace of the alignment used in the GPD method.	205
10.1	A decision rule that governs the routing of process instances at the decision point p_2 of the hospital process.	212
10.2	Decision table for the rules at place p_2 as specified by the DMN standard.	212
10.3	Decision tree learned from a multi-set of training instances.	214
10.4	Overlapping decision rules at decision point p_{10} of the hospital process model.	216
10.5	Overview of the proposed overlapping decision mining method. .	218
10.6	Place fitness and local precision achieved by the proposed method (DTO) compared to the standard decision tree classifier (DTF), and the model without guards (WO).	228
10.7	Average place fitness and place precision achieved by the DTO method compared to the DTT method.	229
10.8	Simplified variant of the process model used for evaluating the decision mining method on the sepsis cases event log.	230
11.1	An overview of the five discovery steps of the iDHM.	240
11.2	The main screen of the iDHM showing conformance information projected on a discovered DC-Net.	241
11.3	Dependency relations determined by the iDHM in its first step. .	242
11.4	Two conditional dependencies highlighted in red, were added in the second step.	242
11.5	Bindings and guarded bindings.	243
11.6	Conformance statistics	243
11.7	DPN converted from the discovered DC-Net by the iDHM tool. The DPN can be used, e.g., for further analysis in the MPE tool.	244
11.8	Main screen of the MPE showing the input process model (base model).	246
11.9	Petri net of the road traffic fine management process used as input to the MPE.	248
11.10	Configuration options of the MPE performance mode	249
11.11	Visualization options of the MPE	250
11.12	Two examples of performance statistics projected on the process model.	251
11.13	Fitness diagnostics projected on the process model.	252
11.14	Precision diagnostics projected on the process model.	253

11.15	Configuration options of the MPE data discovery mode.	253
11.16	Process model with discovered guard expressions for the output transitions of the places $pl10$, $pl12$, and $pl14$ by applying the overlapping decision mining method.	254
11.17	Quality diagnostics for the process model with discovered data perspective in terms of fitness and precision as shown by the MPE.	256
11.18	Trace view of the MPE showing details on the alignment of individual traces to the process model.	257
11.19	Chart view comparing the distribution of values of the attribute <i>expense</i> in the log projection of the alignment for the two output transitions of place $pl14$	258
12.1	The normative DPN created for the road fines management process.	264
12.2	Conformance diagnostics projected on the road fine traffic management DPN as produced by the MPE fitness mode for the balanced alignment method.	267
12.3	Comparison of the <i>MPE fitness mode</i> output returned for the non-balanced method (left) and the balanced method (right).	268
12.4	Comparison between the fitness level computed by the balanced and non-balanced method.	269
12.5	Output of the trace view diagnostics of the MPE for trace σ_A . . .	270
12.6	Output of the trace view diagnostics of the MPE for trace σ_B . . .	271
12.7	Process model enhanced using the overlapping decision mining method.	274
12.8	Quality diagnostics for the enhanced model as shown by the MPE. . .	275
12.9	DC-Net discovered for the fine management event log.	277
12.10	Conformance information projected on the discovered DC-Net. . .	281
12.11	C-Net discovered by the standard Heuristic Miner for the fine management log.	282
12.12	Petri net discovered by the Inductive Miner for the fine management log.	283
12.13	The atomic activity patterns and the composed abstraction model used as input to the GPD method. Both atomic activity patterns were designed based on domain knowledge about the process. . .	284
12.14	Process models discovered by the Inductive Miner when applying GPD.	285
13.1	Definition of sepsis and septic shock as provided by the Surviving Sepsis Campaign	290
13.2	Patient flow and questions as described by the process stakeholders.	291
13.3	Hand-made normative process model for the trajectories of sepsis patients.	295

13.4	Projection of the events on the normative process model as returned by the MPE. There are violations regarding the variable <i>timeAntibiotics</i> are visible (highlighted in orange) and some violations regarding variable <i>timeLacticAcid</i> . For some of the transition a few model moves are diagnosed (highlighted in yellow).	297
13.5	Digital triage form that is filled out in the emergency ward.	298
13.6	DPN discovered by the overlapping decision mining method when applied on the sepsis cases event log and the normative process model without guards.	299
13.7	DC-Net discovered by applying the iDHM to the sepsis cases event log.	301
13.8	Conformance diagnostics obtained by a multi-perspective alignment. The iDHM projects the alignment information onto the DC-Net.	302
13.9	Two manual patterns that were created for the sepsis event log.	304
13.10	Three discovered patterns that were obtained by splitting the log based on the department attribute and using the Inductive Miner on the resulting sub logs.	304
13.11	Abstraction model used for the case study. We added the restriction that the high-level activity Transfer can only occur after the high-level activity Admission.	304
13.12	High-level and expanded Petri net discovered using IM when applying the GPD method. Gray transitions are abstracted high-level activities.	306
13.13	Performance information and a decision rule projected on the expanded model discovered for the sepsis event log.	307
13.14	Unguided Petri net discovered using HM without applying the GPD method.	309
13.15	Unguided Petri net discovered using ILP Miner without applying the GPD method.	309
13.16	Unguided Petri net discovered using ETM without applying the GPD method.	310
13.17	Unguided Petri net discovered using IM without applying the GPD method.	311
14.1	Screenshot of the digital whiteboard software that is used by the Norwegian hospital.	313
14.2	Abstraction model used in the case study. Most activities can only be interleaved (i. e., they are not concurrent) as there is only one nurse assigned to a patient.	315
14.3	Three activity patterns that model high-level activities regarding the patient logistics: Registration, Transfer, and Discharge.	317
14.4	Three activity patterns that model high-level activities in the nurse call system: Shift, Alarm Normal, and Handover.	318

14.5	Three activity patterns for the diagnostic high-level activities: Surgery, CT, and Ultrasound.	318
14.6	Petri net discovered for the low-level event log.	319
14.7	Petri net discovered for the high-level event log.	320
14.8	Output of the MPE fitness model for the Petri net discovered from the high-level event log.	321
14.9	Average time between events projected on a process model modeling the usage of the call signal system.	322
14.10	Chart view of the MPE reveals that some nurses use the <i>quick variant</i> (blue bars) of responding to an alarm instead of the desired variant (red bars) more often than others. The identifiers have been anonymized beforehand.	323
14.11	Dotted charts of events related to the activity <i>Shift</i> . Traces are shown on the y-axis and sorted by the time of day of the first event in a trace.	323
15.1	The <i>desired path</i> through the billing process runs through five states.	325
15.2	Manually created process model for the billing process that we used for visualization purposes and decision mining.	328
15.3	Decision rules discovered for the hospital billing process using the overlapping decision mining method.	329
15.4	Fitness measure projected on both the hospital billing process models without and with decision rules.	332
15.5	Precision measure projected on both the hospital billing process models without and with decision rules.	333
15.6	DC-Net discovered for the hospital billing event log.	335
15.7	Conformance information projected on the discovered DC-Net of the billing process.	337
15.8	Petri net discovered by the Inductive Miner for the hospital billing log. Note that many activities can be skipped and it is possible to loop back. This makes the model very imprecise.	339
15.9	C-Net discovered by the standard Heuristic Miner for the hospital billing log.	339

List of Tables

2.1	Four traces of an event log recorded for the hospital process.	26
4.1	Three alignments between log traces $\sigma_2, \sigma_4 \in \mathcal{E}_{\text{ex}}$ and the hospital process.	61
6.1	Six traces of the event log L_c recorded by the example credit application process.	115
6.2	Precision and fitness scores for the normative and discovered process models.	130
8.1	Three exemplary traces of an event log L_h recorded by the hospital process.	153
9.1	Low-level and high-level activities on the type and instance level. .	179
9.2	Excerpt of an example trace $\sigma_{\text{wb}} \in \mathcal{E}_L$ from an log-level event log L_L that contains low-level events recorded by an electronic whiteboard.	181
9.3	Sources for manual and discovered activity patterns.	187
9.4	Example of an alignment and the resulting abstraction to a high-level log as created by the GPD method.	196
10.1	Excerpts of 4 traces of an event log L_{dec} recorded by the hospital process.	213
10.2	Excerpts of 4 traces of the event log L_{dec}	220
10.3	Guards discovered by the compared approaches at decision point S-p5231	
12.1	Activities recorded in the fine management event log.	262
12.2	Attributes recorded in the fine management event log.	263
12.3	Cost function κ for the fine management process.	266
12.4	Exemplary trace σ_A with an unpaid fine.	269
12.5	Non-balanced and balanced alignment for the log trace σ_A	270
12.6	Exemplary trace σ_B with an underpaid fine.	271
12.7	Non-balanced and balanced alignment for the log trace σ_B	273
12.8	Dependency conditions discovered for the fine management event log.	278
12.9	Decision rules discovered for the guarded bindings.	279

13.1 Activities recorded in the sepsis cases event log.	292
13.2 Attributes recorded in the sepsis cases event log.	293
13.3 Guard expressions encoding the time constraints of the sepsis process.	294
14.1 Activity patterns used for the digital whiteboard case study.	315
15.1 Activities recorded in the hospital billing event log.	327
15.2 Attributes recorded in the hospital billing event log.	327
15.3 Dependency conditions discovered for the hospital billing log. . . .	336
15.4 Decision rules for the billing process DC-Net.	338

List of Algorithms

1	Procedure that computes a balanced alignment	75
2	Procedure that builds a MILP to obtain an optimal variable assignment	84
3	Optimized procedure that computes a balanced alignment	94
4	Procedure building a high-level event log based on an abstraction model and a low-level event log.	198
5	Procedure <i>assignAttributes</i> that assign the attributes of newly created events. Each event represents the execution of a transition in the life-cycle of a high-level activity instance.	199
6	Procedure that discovers the guards of a DPN based on an event log.	221
7	Procedure <i>buildEstimator</i> , which a guard estimator that computes partly overlapping guards.	222

Part I

Introduction

Chapter 1 We introduce multi-perspective process mining and the research problems addressed in this thesis.

Chapter 2 We introduce preliminaries such as basic notations used in this thesis and the notation for event logs.

Chapter 3 We introduce notations for three process modeling languages, which are used in this thesis.

1 Introduction

The efficient and effective handling of its processes is essential for the success of an organization. This thesis is about **process mining**, i.e., analyzing the processes of an organization by using data recorded about their execution. A **process** can be defined as:

a set of interrelated or interacting activities, which transforms inputs into outputs. [ISO15]

A process that is executed in a professional context, is commonly denoted as **business process**: “[...] a set of logically related tasks performed to achieve a defined business outcome.” [DS90] Exemplary business processes are, e.g.:

- the process of handling of a loan application (service),
- the process of treating a patient in the emergency ward (health-care), and
- the process of manufacturing a car (production).

Several tasks or activities are executed in one instance of such a process. A process instance is commonly denoted as a **case**, i.e., the activities of the process operate on the case. Each case of a process has a defined start point and end point. In the remainder of this thesis we use the term *process* but implicitly assume processes to be executed in the context of professional organizations, i.e., processes that describe how cases are handled with a well-defined start and end point. Possible activities that are part of such processes could be, e.g.:

- approving a loan request,
- checking a credit rating,
- filling the triage for a patient,
- taking an X-ray image for medical diagnostics, and
- ordering a missing part.

The seminal articles on business re-engineering by Hammer and Champy [HC93] and Davenport [Dav93] have established the focus on the processes of an organization in management practice: Organizations should radically reorganize their work along their value-adding processes. A large body of work, both from industry and from academia, has been organized around the belief that excellent processes are the foundation of any successful organization. The basic problem that is being tackled is: *How do organizations obtain and execute excellent processes?*

This problem has been addressed from various viewpoints. This resulted in a large body of methods, languages, and tools: For example, management trends

and strategies such as business process re-engineering, lean management, and six sigma¹ and research fields and methods such as workflow management, adaptive case management, and Business Process Management (BPM). Finally, a large number of software systems for process execution has been proposed. For example, Staffware, COSA, YAWL, Bizagi, Bonita, Camunda, jBPM, IBM Business Process Manager, Oracle BPM Suite, and many more, cf. [Mue04, p. 93] for an overview.

Business Process Management (BPM) can be seen as the umbrella-term that encompasses all those methods that are concerned with the *design, enactment, monitoring, and optimization* of processes that handle cases. For all these concerns, in-depth knowledge on the processes of an organization is crucial. This in-depth knowledge is often obtained by manual labor, i. e., consultants observe the process work or conduct interviews with participants of a process to discover what is really been done. However, this is an expensive and slow operation. Moreover, the view of process participants on their own work might often provide a biased view on the processes of an organization.

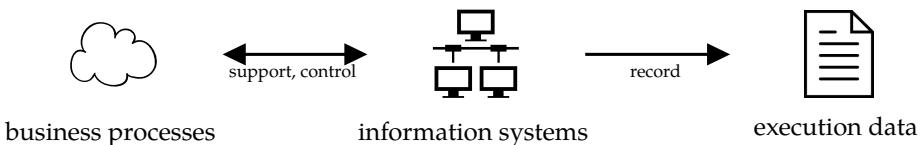


Figure 1.1: Process execution data is recorded for the executions of activities of a process instance. Process mining leverages such data recorded in event logs to analyze the real execution of the process.

Due to the growing computing power and storage capacity of today's IT systems, organizations have the opportunity to **store information about all their activities** that are conducted through information systems. Leveraging knowledge from such recorded data is widely acknowledged to be an important challenge. This is evident through the rise of fields such as data mining, machine learning, artificial intelligence, data science, and big data. Also in information systems research, the challenges of "leveraging knowledge from data, with related management of high data volumes" [Bec+15] has been considered an important grand challenge for IS research. Moreover, experts estimate its solution to have the most impact on the field [Bec+15].

Since most business processes are supported by at least one information system, as depicted in Figure 1.1, the amount of data being stored about process executions is rapidly growing. This data might be recorded by a process-aware information system, e. g., a workflow management system that executes a well-defined process. But also information systems that are not *process-aware* record data about process execution. For example, an Enterprise Resource Planning (ERP) system might be

¹Lean management and Six Sigma both predate the business process re-engineering trend.

used to support the process execution or a purpose-made application might record data about the execution of process instances or cases in log files. Typically, the execution of a case results in a sequence of events (i.e., execution trace) being recorded. In general, such a log trace contains at least:

- the timestamps of activity executions (i.e., **events**), and
- the names or identifiers of the executed activity (i.e., **activity names**).

Process mining leverages such unbiased execution data to *analyze the actual execution of processes* [Aal+12]. Often, process mining methods only make use of activity names and the timestamps of events recorded in execution traces. Other aspects of the process execution are then overlooked. This thesis contributes process mining techniques that make use of *additional data* to analyze a process from *multiple perspectives*.

The structure of this introductory chapter is as follows. First, in Section 1.1 we briefly introduce the foundations of process mining without considering multiple process perspectives. Then, in Section 1.2 we extend our view on process mining towards additional data and multiple process perspectives. In Section 1.3, we formulate the overall research goals and summarize the contributions of this thesis.

1.1 Process Mining

The aim of process mining is to automatically provide an accurate view on how the process is executed. Event logs and process models are two main artifacts that are used in process mining. **Event logs** store data on the actual execution of the cases of a process as recorded by information systems. **Process models** are used as representation of processes.

Event Logs. Process mining methods typically assume that *execution data is stored in event logs*. In event logs, data about each execution of a process (i.e., process instance or case) is recorded as a sequence of events. This sequence of events is denoted as a *log trace*. Each *event* refers to the execution of an activity that was executed as part of

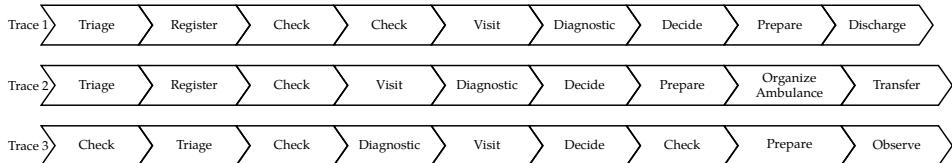


Figure 1.2: Execution traces recorded by two process instances of a hospital process. Events are recorded for each execution of an activity. In both instances the first event refers to an execution of activity *Triage* and the second event refers to an execution of activity *Register*.

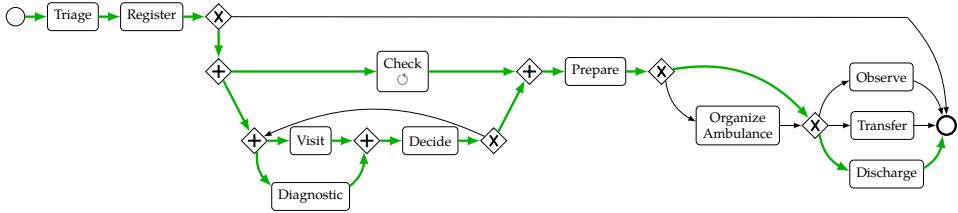


Figure 1.3: The control-flow of an example hospital process modeled using BPMN. The sequence of activity executions recorded in the first trace of Figure 1.2 is highlighted.

the process instance. Figure 1.2 shows three log traces that are recorded for process instances of an example hospital process: 27 events were recorded involving 11 distinct activities. The first two log traces started with an event recorded for the *Triage* activity in which the priority of a patient based on the injuries is determined. The third log trace starts with a *Check* activity. We will elaborate on the remaining activities in Section 1.2.

Process Models. Process models are used to describe, prescribe, and explain [Rol98] the behavior of processes of an organization for a wide range of objectives such as: communication among stakeholders, process improvement, process management, process automation, and process execution support [CKO92]. Concrete examples are the comparison of the as-is and the to-be process, documentation for complying with regulatory requirements such as ISO 9001 [ISO15], and the analysis of performance-related problems such as bottlenecks and inefficiencies. Figure 1.3 depicts the control-flow (i. e., the ordering of activities) of the hospital process using Business Process Model and Notation (BPMN) [BPMN11]. Activities of the process are shown as boxes and the ordering of activities is defined through directed edges and special routing constructs (exclusive choice \times and parallel $+$). Trace 1 from Figure 1.2 is projected on top of the process model. We highlighted the path followed through the model with green color. A comprehensive introduction to the hospital process, which is used as running example throughout this thesis, is provided in Section 1.2.

The field of process mining can be organized in three categories [AAD12; Aal16]: **discovery**, **conformance**, and **enhancement**. Figure 1.4 gives an overview of these main types of process mining in the context of the real process execution (process reality) and information systems of an organization. As shown in Figure 1.4, process mining methods use process models and event logs as proxies for the real execution of processes. An ultimate goal in the field of process mining is the automatic discovery of accurate and understandable process models based solely on the data recorded in an event log. Those models can be used for understanding and improving the real execution of a process.

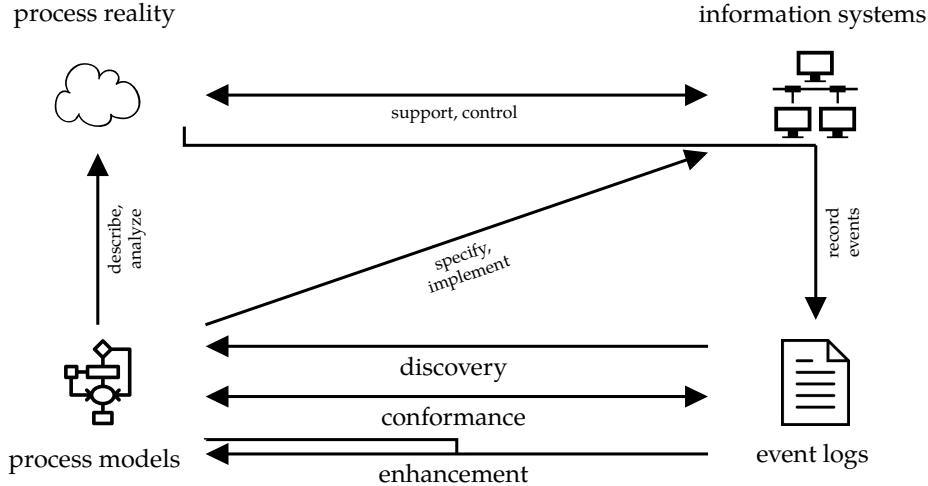


Figure 1.4: Overview of the three main categories of process mining [Aal16].

However, next to process discovery, there are further equally important challenges. We elaborate briefly on the three main categories of process mining. For each category, we list open challenges that are related to the contributions of this thesis. Please note that Figure 1.4 describes process mining only in the offline setting, i. e., only finished process cases are analyzed. Generally, process mining is not limited to the offline setting. It also entails methods such as prediction and recommendation based on current process data in an online setting. In the scope of this thesis, we only consider the offline setting.

Discovery. Process discovery methods solely use the data stored in event logs to automatically generate an accurate process model that describes the real execution of a process. The aim of process discovery is to create process models that:

- describe the observed behavior (i. e., fitting models),
- describe not much more than the observed behavior (i. e., precise models),
- generalize from the exact observed behavior (i. e., general models), and
- are not unnecessarily complex (i. e., simple models).

Given the nature of business processes in part to be based on human behavior, process discovery techniques face some challenges. They need to be able:

- to filter noise (i. e., infrequent and erroneous events) from regular events, and
- to recognize the correct behavioral relations between activities despite an incomplete observation of the process.

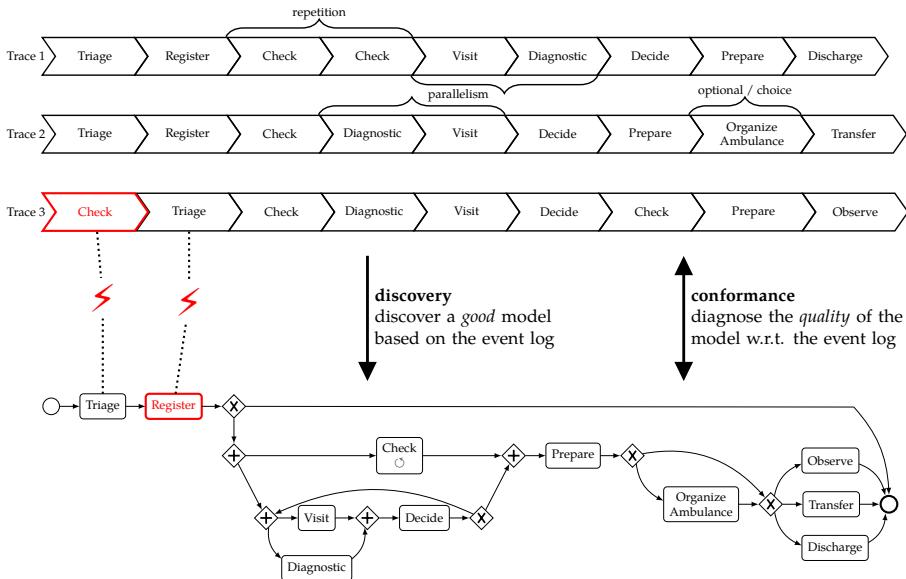


Figure 1.5: Process discovery uses the information stored in the event log to discover a suitable process model. Conformance checking diagnoses deviations between process models and the information stored in the event log.

In Figure 1.5 some of the challenges that process discovery methods face are illustrated using three examples traces and the BPMN model of the hospital process. A good process model needs to be created based on a set of execution traces, which cover only a limited subset of all possible traces. Relations between activities such as sequence (e.g., *Triage* and *Register*) repetition (e.g., *Check* activity), parallelism (e.g., *Visit* and *Diagnostic* activities), and choice (e.g., the optional *Organize Ambulance* activity) need to be discovered. However, noise and incompleteness of the event log need to be considered. For example, the first occurrence of activity *Check* before *Triage* could be considered as noise since this activity does not occur before *Triage* in the first two traces. Moreover, the event log is probably incomplete. It is not certain whether activity *Check* can be repeated an unlimited number of times and whether activity *Check* occurs in parallel to the activities *Diagnostic* and *Visit*.

Conformance. Conformance checking methods provide diagnostic information and quantification of discrepancies between the actual process execution and a discovered or manually created process model. Some of the challenges that conformance checking methods face are [Aal+12]:

- to relate process model elements to events,
- to determine reliable quality measures for process models,

- to balance the trustworthiness of the event data and the process model, and
- to provide reliable and understandable diagnostics.

In Figure 1.5 some of challenges that conformance checking methods face are illustrated. Conformance checking methods aim to relate each event in the event log to a corresponding element in the model. For example, the first event in the third trace records an execution of activity *Check*. However, this event cannot be related to the activity *Check* as modeled in the process model, since the activity is not allowed to be executed at the start of a case. It is also possible that no events can be found for activities in the model, e. g., in the third trace the event for activity *Register* is missing. Conformance checking methods diagnose, amongst other tasks, such discrepancies.

Enhancement. Enhancement methods use the recorded execution data to improve *existing process models*. Often, process models exists as part of the process documentation or, the basic control-flow of process models was discovered by a process discovery method. These process models can be extended with information based on the process context, e. g., decision logic, performance indicators, queuing models. Process models can also be repaired based on conformance checking results in order to better reflect the real process execution. One of the challenges that enhancement methods face is to avoid the curse of dimensionality when considering data recorded in the process context [Aal+12]. In Figure 1.5, enhancement methods could enrich the model, e. g., with the conditions under which the optional activity *Organize Ambulance* is executed.

1.2 One Process – Multiple Perspectives

In this section, we extend our view on processes and process mining towards **multiple perspectives**. We introduce the scope of this thesis and identify five concrete perspectives that we considered in our research. So far, we have considered a very simplistic view on the processes of an organization. In Figures 1.2 and 1.3 we assumed that:

- events only record the fact that an atomic activity has been executed, and
- process models only describe the order of activity executions.

Whereas this simplification of process reality can be a “[...] a purposeful abstraction of the behavior [...]” [Aal+12], often, there is more complexity to the real execution of a process. As it is stated in the process mining manifesto [Aal+12]: “Process mining is not limited to control-flow discovery.” Process discovery, conformance, and enhancement methods should take advantage of additional data and consider additional perspectives on the process.

1.2.1 Multi-perspective Process Models and Event Logs

We start by illustrating the considered type of input and output: events logs enriched with data attributes and multi-perspective process models. Event logs typically contain more information than just timestamps and activity names. They can contain:

- identifiers of resources that execute an activity (e.g., humans, machines),
- input data used to execute an activity (e.g., patient age, loan amount),
- output data generated by activity executions (e.g., decisions, outcomes), and
- information on the relation between multiple events (e.g., activity lifecycles).

Moreover, real-life activities rarely are atomic constructs. Often, there is a hierarchy of activities: multiple activities executed together form an activity on a higher level of abstraction. Furthermore, process models define more than just the ordering of activities. Often, rules based on data associated to the process instance and contextual information are included, e.g.:

- patients are only admitted to the emergency ward if their triage priority is high (decision rule), and
- the background check for a credit application should be done by a different person than the resource handling the application (four-eyes principle).

For example, the Decision Model and Notation (DMN) [DMN16] standard, a novel standard for managing such decision rules has been recently endorsed by BPM vendors such as Camunda and Signavio.

We are now refining the description of the hospital process that is used as a running example throughout this thesis. At this stage, we describe the process by natural language statements since we do not want to focus on any particular process modeling notation. The process has been deliberately simplified such that it is easy to understand and retains enough details to illustrate our contributions.

Example 1.1 (Description of the hospital process). The process *starts* when patients arrive at the emergency ward of an hospital. Upon their arrival, patients are assigned a *triage color*. Only in exceptional cases, patients are assigned the triage color *white*. Patients classified as *white* typically *leave* the emergency ward after being registered, because their injuries do not require an urgent, immediate attendance by a doctor. All other patients are also *registered*, assigned to a responsible nurse, and admitted to the emergency ward. While patients are in the emergency ward, the nurse *checks* their condition every hour. For the patients under consideration the medical examination consists of at least one medical *diagnostic test* and one *visit* by a doctor. There are two different work practices regarding these two activities:

1. Normally, a doctor first visits the patients and *afterwards* the diagnostic

test is conducted.

2. Sometimes, these activities are executed in a *reversed order*: first the medical diagnostic test is taken and, only thereafter, a doctor visits the patient.

Both the medical diagnostic test and the visit of a doctor can be repeated if necessary. Afterwards, a doctor visits the patient one more time and decides how to proceed in any of the following ways:

1. to transfer the patient to a ward *within* the hospital,
2. to transfer the patient to a *another* hospital (tertiary care), or
3. to *discharge* of the patient.

Regardless of the decision, the patient is prepared for a possible transfer or discharge. If possible, the hospital wants to implement the *retain familiar* constraint. The nurse who registered the patient shall also prepare the patient for transfer or discharge. For a specific group of patients, i. e., those who are transferred to another hospital, an *ambulance* needs to be organized. Finally, the patient leaves the emergency ward of the hospital, either by being transferred, being discharged, or being moved to a special observatory ward for further observation. Patients that are moved to the observatory ward may be subject to further examinations, which we consider out of scope of this process.

The process model in BPMN notation shown in Figure 1.3 describes the same possible ordering of activities as the textual description in Example 1.1. However, the textual description in Example 1.1 specifies information from additional perspectives on the process execution. Each perspective refers to a particular aspect of the process. The model in Figure 1.3 describes the *control-flow perspective* of the process, i. e., the dynamic behavior of the process expressed by the possible orderings of activities for a single process instance. Next to the control-flow perspective, processes can be considered from several other perspectives. Five of these are introduced in the next section.

1.2.2 Five Considered Perspectives

Figure 1.6 depicts five perspectives on the running example process that are often considered in the literature on BPM, process modeling, and process mining [Aal+12; Aal16; BMS16; CKO92; JB96; LA13a; RAH16; Ram17; Ros+11; Sch00]: the control-flow perspective, the resource perspective, the data perspective, the time perspective, and the function perspective. This set of perspectives is not comprehensive and there may be other or additional perspectives from which processes can be looked upon, e. g., costs or risks. However, we argue that these five perspectives are significant perspectives for process mining.

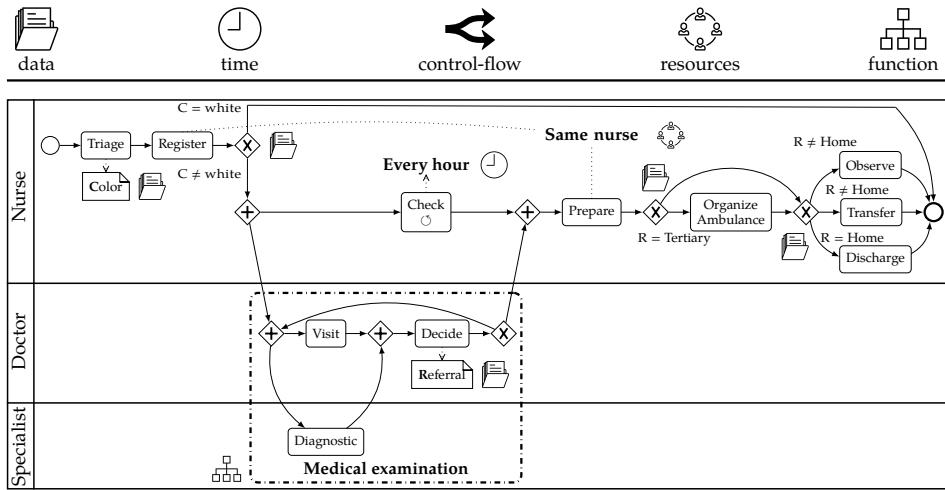


Figure 1.6: Multiple perspectives on a process can be used for process mining. The control-flow perspective (i.e., the ordering of activities) can be combined with other perspectives such as resources, data, time and the hierarchy of functions.

Control-flow perspective. The control-flow perspective, sometimes also called the behavior- or behavioral perspective, of a process describes the order in which its activities should be executed. The overall control-flow of a process corresponds to all the possible sequences of activities. In Example 1.1, it is defined that the patient is first triaged and only afterwards registered, i.e., activities *Triage* and *Register* in the BPMN model Figure 1.3 are depicted in sequence. A second example is that according to Example 1.1 there is no specific control-flow constraint defined regarding the activity *Check* and the activities *Diagnostic* and *Visit*. Therefore, checking the patients may be done in parallel to these activities. The control-flow perspective constitutes the foundation of a process model. Therefore, the control-flow perspective is, usually, the starting point for a process mining analysis [Aal16].

Resource perspective. The resource perspective, sometimes also called the organizational perspective, describes the resources required for the execution of a process and how they interact with each other. Resources can be human resources and non-human resources (e.g., machines and materials). Possible artifacts of the resource perspective can be, e.g., social network graphs, assignment rules, and allocation constraints regarding which resources may execute activities. In Example 1.1 some activities are executed by nurses and some other activities are executed by doctors. In Figure 1.6, BPMN lanes are used to express this allocation rule. Moreover, a *retain familiar resource* constraint is defined for activities *Register* and *Prepare*. Both activities should be conducted by the same nurse. Since the BPMN standard does

not define a specific symbol for such a constraint, we illustrated the constraint in Figure 1.6 by adding an annotation that is connected to both activities.

Data perspective. The data perspective, also denoted as case-, object-, information- or informational perspective, describes which existing data objects are required as input during the execution of the process, used for control-flow routing decisions, and how data objects are created and updated during the execution of the process. In Example 1.1 a triage color is recorded for each patient. This color is used for the routing decision on whether patients need to leave the hospital or are admitted to the emergency ward. In Figure 1.6, we added the data object *Color* to the BPMN process model. The value of this data object is written by the *Triage* activities. Later in the process the recorded value is used to route process instances according to the described rule. Similarly, we added routing decision rules and data object concerning the referral of patients.

Time perspective. The time perspective focuses on all time-related aspects of the process. In addition to the ordered sequence considered by the control-flow perspectives, activity executions take time and occur at a specific moment in time, e. g., before a predefined deadline. Often, there are rules regarding the timing between activities too. In Example 1.1, a nurse needs to check the patient every hour. We added an annotation to the activity *Check* of the BPMN model in Figure 1.6 to express this constraint.

Function perspective. The functional perspective is concerned with the activities that are part of the process. Often, not all activities (i. e., functional units) of a process are at the same abstraction level. Often, the execution of a series of activities at a low abstraction level together form an activity at a higher level of abstraction. Some high-level activities may, in fact, be complex sub processes. For example, in Figure 1.3 the activities *Diagnostic*, *Visit*, and *Decide* can be seen together as activity *Medical examination* at a higher level of abstraction. We expressed this in the BPMN model by annotating the group of activities with a dashed line.²

1.2.3 Relation to Existing Multi-perspective Frameworks

We briefly show how the perspectives considered in other research fields relate to the five perspectives that are considered in this thesis.

A division of the description of the architecture of an entire organization in multiple perspectives is commonly made by Enterprise Architecture Modeling (EAM) frameworks to reduce complexity. EAM frameworks such as the Zachman framework, the CIMOSA framework, and the ARIS framework [Sch00; Sch92] typically

²We did not use the sub process notation of BPMN since we want to keep the example limited to one process diagram.

describe an organization from multiple perspectives. For example, the Zachman framework makes use of the perspectives data, function, network, people, time, and motivation [Zac87]; the CIMOSA framework includes four views on the organization: function, information, resource, and organization [ESP93; KZ99]; and the ARIS framework uses the views organization, data, control, function, and product/service.

We relate our usage of the term *perspective* in the context of those definitions made by EAM frameworks. We use ARIS as an example since it is one of the most widely used EAM frameworks. Our five-perspective view on processes is inspired by the perspectives that ARIS and the other EAM frameworks introduce. Similar to the ARIS methodology we consider the links between the static views to the behavioral model (i.e., the control-view) to be important. We assume that our information originates from an event log that records information about the dynamic process behavior. Thus, we only consider those parts of the resource-, time³, data-, and function perspective that influences the control-flow of the process. This is different from ARIS, which can take a more holistic view because it is a conceptual modeling framework rather than an automated process mining technique.

To conclude, the term *multi-perspective* as used in this thesis relates to the observation that multiple perspectives on processes are connected and considering multiple perspectives together provide a more comprehensive view.

1.3 Research Goals and Contributions

In this thesis we only consider perspectives that are *intertwined with the control-flow perspective*, i.e., there are dependencies between the control-flow of a process and the perspective. Therefore, we do not aim to consider one of the perspectives in isolation, e.g., by discovering a social network graph without considering the control-flow. We target problems in which *multiple perspectives on a process are viewed together*, e.g., data objects that influence the routing of activities, routing that influences the possible resources, routing that depends on time constraints (e.g., fast vs. normal procedure).

Based on this premise, we refine the overall goals of our research. We will also describe the contributions achieved.

1.3.1 Overall Research Goals

In this thesis, we focus on three major research goals. We discuss each of the goals and relate them to the challenges stated by the process mining manifesto [Aal+12].

³The time perspective is not explicitly mentioned in the ARIS framework. However, it would fit most naturally in the control view of ARIS.

Goal 1: Development of process mining methods that consider the interaction between multiple process perspectives. We want to develop discovery, enhancement, and conformance checking methods that consider the **interaction of multiple perspectives on the process**. We aim to advance the use of multi-perspective information for all three types of process mining instead of focusing on one specific type. Moreover, the goal is to consider situations in which perspectives interact with each other, e.g., the choice of resources affects the control-flow. This research goal is related to challenge C5, “Improving the Representational Bias Used for Process Discovery” [Aal+12], of the process mining manifesto. Considering multiple perspectives on a process requires suitable representations, e.g., using a process modeling notation that allows to capture the perspectives. Moreover, it is also related to challenge C6, “Balancing between Quality Criteria Such as Fitness, Simplicity, Precision and Generalization” [Aal+12], of the process mining manifesto. There is a need to reliably determine the quality criteria for multi-perspective process models.

Goal 2: Implementation of efficient and effective tools that can deal with realistic event logs. Often, research prototypes are implemented just as proof-of-concept tools, which can only be used in a very small set of cases. The implementation of more broadly usable tools entails many challenges and involves considerable effort. However, efficient, effective and usable tools are essential to facilitate the adoption of research results in practice. Therefore, one of our research goals is **the development of tools that can deal with realistic event logs in an efficient and effective manner**. This research goal is related to the challenges C1, C10, and C11 of the process mining manifesto [Aal+12]. The challenges are “Finding, Merging, and Cleaning Event Data” [Aal+12] (C1), “Improving Usability for Non-Experts” [Aal+12] (C10), and “Improving Understandability for Non-Experts” [Aal+12] (C11).

Goal 3: Applicability of the method in real-world scenarios. We aim that our methods are **applicable in real-world scenarios**. Therefore, the evaluation of the proposed methods is conducted with four extensive case studies using real-life data. This goal requires that developed methods can deal with the size and the complexity of real-life data. This research goal is related to challenge C2 of the process mining manifesto: “Dealing with Complex Event Logs Having Diverse Characteristics” [Aal+12].

1.3.2 Contributions

We categorize our five main contributions along the three main types of process mining: *conformance*, *enhancement*, and *discovery*. Furthermore, we present the *implementation* and the *application* in real-life situations of all proposed methods as two additional, orthogonal contributions.

Conformance. We contribute the following two multi-perspective conformance checking methods.

- A method to compute an alignment of a multi-perspective process model to an event log where the deviations with regard to the different perspective are given the same importance (Chapter 5). The method can be used for conformance checking of multi-perspective process models and provides reliable diagnostics and quality measures with respect to all perspectives of the process model.
- A method to compute the precision quality measure for multi-perspective process models based on an alignment (Chapter 6). The precision score acknowledges the added precision of decision rules, resource constraints, and time constraints.

Discovery. We contribute the following two multi-perspective process discovery methods.

- The Data-aware Heuristic Miner (DHM) (Chapter 8), a multi-perspective process discovery method that uses the data perspective (i. e., recorded data attributes) to distinguish infrequent paths from random noise by using classification techniques. Data- and control-flow are learned together, i. e., recorded data values are used to build improve the discovered control-flow.
- The Guided Process Discovery (GPD) method (Chapter 9), a process discovery method that uses domain knowledge expressed as multi-perspective activity patterns to abstract low-level activities to high-level activities (i. e., considers the function perspective). Grouping low-level events to recognizable activities on a higher abstraction level helps to discover a process model that can be understood by stakeholders.

Enhancement. Regarding the enhancement category of process mining, we contribute a method to discover potentially overlapping decision rules in process models based on an event log (Chapter 10). Overlapping (i. e., non mutually-exclusive) decision rules are often encountered in practice since business rule may be non-deterministic and contextual information relevant for the actual decision making is unavailable. The method balances precision and fitness of a process model with regard to an event log. When rules are overlapping two or more possible routing options can be chosen non-deterministically. As result, the process model is less precise but fits the observations better.

Implementation. We implemented all proposed methods in the open source framework ProM in the form of plug-ins. Moreover, we integrated the functionality in two interactive tools: the *Multi-perspective Process Explorer* (MPE) and the *Interactive*

Data-aware Heuristic Miner (iDHM). Both tools reached a high level of maturity and have been used in several real-life situations.

Applications. We applied all proposed methods in the context of four case studies conducted in several organizations. For each case study, we obtained real-life event data from their information systems, identified process questions relevant to the organization, and showed that the application of our methods is *feasible* and provides *valuable insights*. Note that the focus of the conducted case studies was on the evaluation of the proposed methods in a real-life environment and not on solving a business questions. Thus, we only sketch the business context of each case briefly and restrict ourselves to applying the proposed methods instead of conducting a full case study which would be focused on the business questions.

1.4 Structure

This thesis is structured in five parts and 16 chapters. An overview of the structure is given in Figure 1.7.

Part I: Introduction. The introductory Chapter 1 briefly provides the necessary context to understand the contributions that are made in this thesis. Chapter 2 provides the preliminaries such as mathematical notations and formal definitions of event logs. Chapter 3 describes the process model notations that are used in the remainder of this thesis.

Part II: Multi-perspective Conformance. Chapter 4 introduces conformance checking and multi-perspective conformance checking in more detail and introduces basic concepts such as alignments. Chapter 5 describes a balanced alignment method for multi-perspective process models. Chapter 6 introduces a precision measure for multi-perspective process models based on multi-perspective alignments.

Part III: Multi-perspective Enhancement and Discovery. Chapter 7 introduces multi-perspective process discovery and enhancement in more detail. Chapter 8 introduces a discovery method that considers the data perspective as recorded in the event log to improve discovery of the control-flow perspective. Chapter 9 provides a method for process discovery that is guided by multi-perspective activity patterns, which represent high-level activities of the process. Chapter 10 describes an enhancement method to discover overlapping decision rules for process models.

Part IV: Applications. In Chapter 11 we describe the implementation of the introduced methods as part of the open source process mining framework ProM.

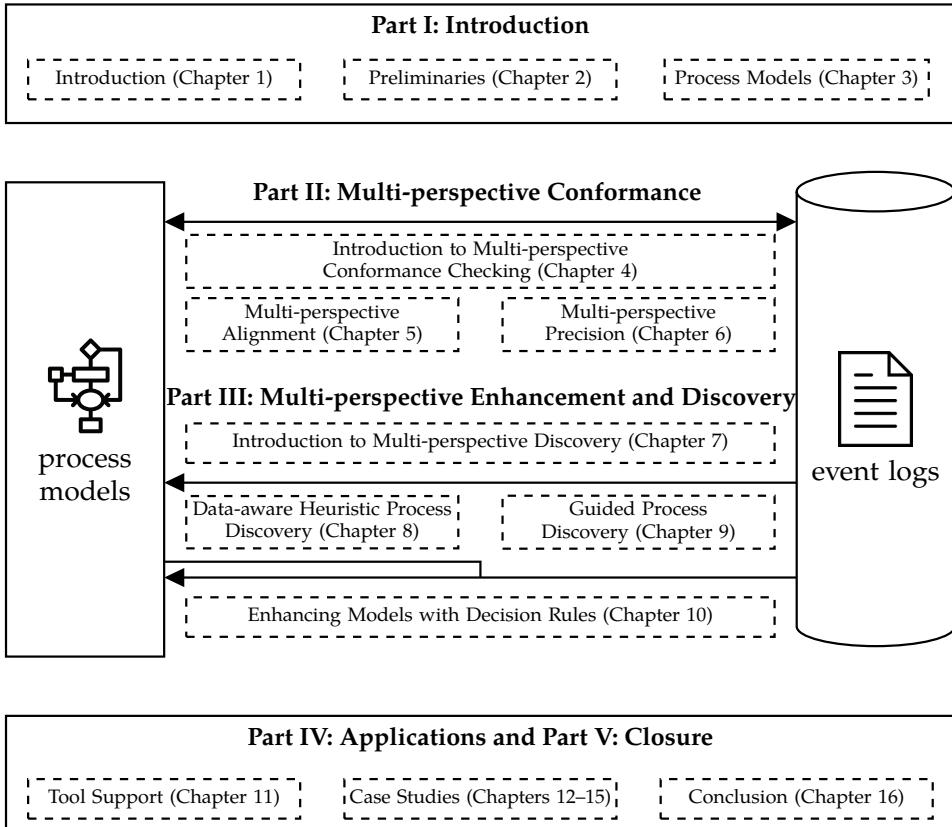


Figure 1.7: Structure of this thesis in context of the three types of process mining.

Specifically, we present two interactive tools that have been developed: the Interactive Data-aware Heuristic Miner (11.1) and the Multi-perspective Process Explorer (11.2). Then, we present four case studies in Chapters 12 to 15. In each of the case studies we tested our proposed methods in real-life situations.

Part V: Closure. Chapter 16 concludes with summarizing our contributions and elaborating on future work.

2 Preliminaries

In this chapter, we introduce necessary preliminaries such as basic mathematical notations, expressions, decision tree classifiers, and the notation for events logs.

2.1 Basic Notations

Definition 2.1 (Set). A set is an unordered collection of distinct objects, which are called *elements* of the set. A set is *finite* if it contains a finite number of elements. Otherwise, the set is *infinite*. We describe a finite set by listing each element of the set between curly braces, e.g., $X = \{a, b, c\} = \{c, b, a\}$ is the set with elements a, b, and c. We use symbol \in to denote that an object is contained as element in a set, e.g. for set X with elements a, b, and c, $a \in X$ whereas $d \notin X$. \diamond

Infinite sets can be described by using an ellipsis, e.g. $\mathbb{N} = \{0, 1, 2, 3, \dots\}$ is the set of all natural numbers, or the set-builder notation, e.g., $\{n \in \mathbb{N} \mid \exists k \in \mathbb{N} : n = 2k\}$ is the set of even numbers. Let X and Y be two sets. We use the following standard notations for sets:

- \emptyset denotes the empty set, i.e., $\emptyset = \{\}$,
- $|X| = n$ denotes the number of elements (*cardinality*) of a finite set X,
- $X \cup Y$ is the *union* of X and Y,
- $X \cap Y$ is the *intersection* of X and Y,
- $X \setminus Y$ is the *difference* of X and Y,
- $X \subseteq Y$ denotes that X is a *subset* of Y,
- $X \subset Y$ denotes that X is a *strict subset* of Y,
- $\mathcal{P}(X) = \{Y \mid Y \subseteq X\}$ denotes the *power set*, i.e., the set of all subsets over X, and
- $X \times Y = \{(x, y) \mid x \in X \wedge y \in Y\}$ denotes the *cartesian product* of X and Y. Elements (x, y) are ordered pairs (i.e. tuples).

Definition 2.2 (Relation). Let X_1, X_2, \dots, X_n be sets. A n-ary *relation* R over sets X_1, X_2, \dots, X_n is a subset of the cartesian product of the sets, i.e., $R \subseteq X_1 \times X_2 \times \dots \times X_n$. \diamond

Relations define a mapping between elements of multiple sets. Functions are special relations, which defines a unique mapping from elements of a source set to elements of a target set.

Definition 2.3 (Function and partial function). Let X and Y be sets.

- A binary relation $f \subseteq X \leftrightarrow Y$ is called a *partial function* if and only if it maps elements in its domain X to *unique elements* in Y :

$$\forall_{x \in X} \forall_{y \in Y} \forall_{\bar{y} \in Y} ((x, y) \in f \wedge (x, \bar{y}) \in f) \implies y = \bar{y}.$$

- A binary relation $f \subseteq X \rightarrow Y$ is called *function* if and only if f maps *every element* in its domain X to unique elements element in Y :

$$\begin{aligned} \forall_{x \in X} \forall_{y \in Y} \forall_{\bar{y} \in Y} ((x, y) \in f \wedge (x, \bar{y}) \in f) &\implies y = \bar{y} \\ \wedge \forall_{x \in X} \exists_{y \in Y} ((x, y) \in f). \end{aligned}$$

- Function $f : X \rightarrow Y$ is surjective if $\forall_{y \in Y} \exists_{x \in X} (f(x) = y)$.
- Function $f : X \rightarrow Y$ is injective if $\forall_{x \in X} \forall_{\bar{x} \in X} (f(x) = f(\bar{x})) \implies x = \bar{x}$.
- Function is bijective if it is both surjective and injective. \diamond

We introduce the following additional notation for partial functions and functions.

- The *domain* of a (partial) function $f : X \leftrightarrow Y$ is denoted as $\text{dom}(f) = \{x \in X \mid (x, y) \in f\}$ and its *range* as $\text{rng}(f) = \{y \in Y \mid (x, y) \in f\}$.
- We denote with $\emptyset : \emptyset \rightarrow \emptyset$ the *empty function* for which the domain and range are empty.
- We denote with $(x_1 \mapsto y_1, \dots, x_n \mapsto y_n)$ an *anonymous partial function*, i. e., the mapping defines a partial function f with $f(x_i) = y_i$ for all $1 \leq i \leq n, x_i \in X, y_i \in Y$. We use this notation when X and Y are clear from the context.
- We denote with $f_1 \oplus f_2$ the *overriding union of functions* f_1 and f_2 :

$$\begin{aligned} \text{dom}(f_1 \oplus f_2) &= \text{dom}(f_1) \cup \text{dom}(f_2) \\ \text{and} \end{aligned}$$

$$\forall_{x \in \text{dom}(f_1 \oplus f_2)} \left((f_1 \oplus f_2)(x) = \begin{cases} f_2(x) & \text{if } x \in \text{dom}(f_2) \\ f_1(x) & \text{otherwise.} \end{cases} \right)$$

Next to sets and functions, the concept of a *multiset* is used throughout this thesis. Multisets generalize sets by allowing the repetition of elements within the set.

Definition 2.4 (Multiset). Let X be a set. A multiset (bag) M is a tuple $M = (X, m)$ where X is the underlying set and $m : X \rightarrow \mathbb{N}$ is the multiplicity function of the multiset. We use $x \in M$ to denote that element x is contained in the multiset M , i.e., $x \in X$ and $m(x) \geq 1$. We denote with $\mathbb{B}(X)$ the set of all multisets over X . \diamond

Often, we use a compact notation for multisets. For example, we write $M = [a, a, b]$ or $M = [a^2, b]$ for the multiset $M = (\{a, b\}, m)$ with $m(a) = 2$ and $m(b) = 1$. We define the following notation and operations on multisets M_1 and M_2 .

- The sum of two multisets is denoted as $M_1 \uplus M_2 = (X_1 \cup X_2, m)$ with

$$m(x) = \begin{cases} m_1(x) + m_2(x) & \text{if } x \in (X_1 \cap X_2) \\ m_1(x) & \text{if } x \in (X_1 \setminus X_2) \\ m_2(x) & \text{if } x \in (X_2 \setminus X_1). \end{cases}$$

- The difference of two multisets is denoted as $M_1 \setminus M_2 = (X_1, m)$ with

$$m(x) = \begin{cases} \max(0, m_1(x) - m_2(x)) & \text{if } x \in (X_1 \cap X_2) \\ m_1(x) & \text{if } x \in (X_1 \setminus X_2). \end{cases}$$

- We denote with $M_2 \leq M_1$ that M_2 is a sub multiset of M_1 , i.e., $M_2 \leq M_1 \iff \forall_{x \in X_2} (x \in X_1 \wedge m_1(x) \leq m_2(x))$.

All operations on multisets can also be applied to a mix of sets and multisets, i.e., sets X can be seen as multisets (X, m) in which each element occur only once.

Example 2.1 (Operations on multisets). Given two multisets $M_1 = [a, b]$ and $M_2 = [b, c]$, and a set $X = \{c\}$. Then, $M_1 \uplus M_2 = [a, b^2, c]$, $M_1 \uplus X = [a, b, c]$, and $M_2 \uplus X = [b, c^2]$. Moreover, $M_1 \setminus M_2 = [a]$, $M_2 \setminus M_1 = [c]$, and $M_2 \setminus X = [b]$. Finally, we have $X \leq M_2$, $M_1 \leq M_1 \uplus M_2$ and $M_2 \leq M_1 \uplus M_2$.

Sequences are used widely in this thesis. Sequences are ordered collections of elements. The same element can appear multiple times.

Definition 2.5 (Sequence). Let X be a set. A non-empty finite sequence s of length n over elements in X is a function $\sigma : \{1, 2, \dots, n\} \rightarrow X$. Function σ defines in which order elements of set X appear. We denote a sequence using the notation $\sigma = \langle s_1, \dots, s_n \rangle$ where $s_i = \sigma(i)$, for $1 \leq i \leq n$. We denote with $s_i \in \sigma$ that element s_i is element of the sequence σ and appears at the i -th position. The *empty sequence* with length 0 is denoted with $\langle \rangle$. A *finite sequence* is either the empty sequence or a non-empty sequence of a certain length. \diamond

The set of all finite sequences over X is obtained through the Kleene star operator X^* . A sequence $\sigma \in X^*$ is also called *word* and the set X the *alphabet*. Let $\sigma_a = \langle a_1, \dots, a_n \rangle$ and $\sigma_b = \langle b_1, \dots, b_n \rangle$ be sequences over X .

We define the following five operations on sequences.

- We denote with $|\sigma| \in \mathbb{N}$ the *length* of a sequence, i.e., $|\langle s_1, \dots, s_n \rangle| = n$.
- We denote with $\sigma_a \cdot \sigma_b \in S^*$ the *concatenation* of two sequences, i.e., $\sigma_a \cdot \sigma_b = \langle a_1, \dots, a_n, b_1, \dots, b_n \rangle$. It is also possible to concatenate single elements to sequences, i.e., we write $\sigma_a \cdot x = \sigma_a \cdot \langle x \rangle$ to append $x \in X$ to sequence σ_a .
- We define $\sum_{s \in \sigma} f(s) = f(s_1) + \dots + f(s_n)$ as the sum of function $f : X \rightarrow \mathbb{N}$ applied to all elements of the sequence σ .

- Given a subset $Y \subseteq X$, we denote with $\text{proj}(\sigma, Y)$ the *projection* of $\sigma = \langle s_1, \dots, s_n \rangle$ on Y , i.e., $\text{proj}(\sigma, Y) \in Y^*$ is obtained by removing all elements $s_i \notin Y$ for $1 \leq i \leq n$ from sequence σ .
- Given a sequence σ , we denote with $\text{prefix}(\sigma)$ the set of all prefixes of σ , i.e., $\sigma_{\text{pre}} \in \text{prefix}(\sigma)$ if and only if a sequence σ_{suf} exists such that $\sigma_{\text{pre}} \cdot \sigma_{\text{suf}} = \sigma$.

Example 2.2 (Sequences and operations on sequences). Given the set of possible elements $X = \{d, e, o, r, w\}$, we can build sequences based on set X . For example, $\langle w, o, r, d \rangle \in X^*$ and $\langle d, o, o, r \rangle \in X^*$ are sequences. We can combine two sequences, by concatenation to a new sequence, e.g., $\langle w, o \rangle \cdot \langle r, d \rangle = \langle w, o, r, d \rangle$ and $\langle w, o \rangle \cdot w = \langle w, o, w \rangle$. Given a function f that maps each of the elements of X to a number, e.g., $f(w) = f(o) = f(r) = 1$ and $f(d) = 2$, we can obtain the sum of f applied to sequence $\langle w, w, d \rangle$ as $\sum_{s \in \langle w, w, d \rangle} f(s) = 1 + 1 + 2 = 4$. We can project a sequence on a subset of the alphabet X , e.g., $\text{proj}(\langle w, o, r, d \rangle, \{o, r\}) = \langle o, r \rangle$ and $\text{proj}(\langle d, o, o, r \rangle, \{o\}) = \langle o, o \rangle$. The set of all prefixes of $\langle w, o, r, d \rangle$ is $\text{prefix}(\langle w, o, r, d \rangle) = \{\langle \rangle, \langle w \rangle, \langle w, o \rangle, \langle w, o, r \rangle, \langle w, o, r, d \rangle\}$.

In the remainder of this thesis, as a matter of convention, we use upper case identifiers to denote sets and multisets (e.g., $X = \{a, b, c\}$); lower case identifiers to denote functions (e.g., $f : X \rightarrow Y$) and elements of sets (e.g., $a \in X$); and bold symbols to denote sequences (e.g., σ).

2.2 Variables and Guard Expressions

Variables and their values are used to hold the values of data objects that are created, updated, recorded, and read during the execution of a process.

Definition 2.6 (Variables). Let U be an universe of values. We denote with V the set of all variables. Variables are defined over a (potentially) infinite domain of values. Given a variable $v \in V$, we use function $\text{dom}(v) \in \mathbb{P}(U)$ to retrieve the domain of the variable.⁴ ◊

During a process execution a set of variables may be associated with concrete variable values. We denote a function that captures a concrete assignment of variable values as variable assignment.

Definition 2.7 (Variable Assignments). Let $V_X \subset V$ be a set of variables. A variable assignment $w \in V_X \rightsquigarrow U$ is a partial function that assigns values $u \in U$ to variables $v \in V_X$. The set of all valid variables assignment for variables in V_X

⁴Function dom is overloaded depending on the context. When applied to functions it returns their domain. When applied to variables it returns the domain of admissible values.

according to their domain dom is:

$$\mathcal{U}_{dom}(V_X) = \{w \in V_X \rightarrow U \mid \forall_{v \in dom(w)}(w(v) \in dom(v))\}. \quad \diamond$$

To simplify the notation, we abbreviate $\mathcal{U}_{dom}(V_X)$ as \mathcal{U}_{dom}^X in the remainder of this thesis.

We use *linear boolean expressions* to formulate constraints over multiple perspectives of a process that can be represented as variables.

Definition 2.8 (Linear boolean expression). Let V be a set of variables. We denote the universe of all *boolean expressions* over variables V as $EXPR(V)$. A linear guard expression is a *boolean formula* $expr \in EXPR(V)$ that evaluates to true or false. We define a concrete syntax for guard expressions using the following grammar in EBNF notation:

```

⟨orExpr⟩      ::= ⟨andExpr⟩ ((‘∨’ ⟨andExpr⟩))*
⟨andExpr⟩    ::= ⟨eqTerm⟩ ((‘∧’ ⟨eqTerm⟩))*
⟨eqTerm⟩     ::= ⟨relTerm⟩ ((‘=’ ⟨relTerm⟩) | (‘≠’ ⟨relTerm⟩))*
⟨relTerm⟩    ::= ⟨addTerm⟩ ((‘<’ ⟨addTerm⟩) | (‘≤’ ⟨addTerm⟩)
                      | (‘>’ ⟨addTerm⟩) | (‘≥’ ⟨addTerm⟩))*
⟨addTerm⟩    ::= ⟨mulTerm⟩ ((‘+’ ⟨mulTerm⟩) | (‘−’ ⟨mulTerm⟩))*
⟨mulTerm⟩    ::= ⟨unTerm⟩ ((‘.’ ⟨unTerm⟩) | (‘÷’ ⟨unTerm⟩))*
⟨unTerm⟩     ::= (‘−’ ⟨unTerm⟩) | (‘≠’ ⟨unTerm⟩) | ⟨variable⟩ | ⟨literal⟩ | (‘(’ ⟨orExpr⟩ ‘)’)
⟨variable⟩   ::= (⟨name⟩ [“”])
⟨literal⟩    ::= ⟨int⟩ | ⟨float⟩ | ⟨string⟩ | ‘true’ | ‘false’ | ‘⊥’

```

The non-terminals $\langle int \rangle$, $\langle float \rangle$, and $\langle string \rangle$ are defined in the standard manner. The non-terminal $\langle name \rangle$ is a special string literal denoting a variable. Additionally we require *linear boolean expressions* to fulfill several requirements, which we list separately from the syntax to avoid an overly verbose grammar.

- Arithmetic terms are limited to linear terms, i.e., each term may only be constant or the product of a constant with a single variable (e.g., $x \cdot x$ is *not* a linear term).
- Literals and variables from domains with incompatible equivalence relations may not be used in one term (e.g., “test” $<$ 10 is undefined, since $<$ is not defined for both strings and integers together).
- Literals and variables of type boolean and string may not be used with arithmetic terms ($+, -, \cdot, \div$).
- Variables of type boolean may not be used with relational terms ($<, \leq, >, \geq$).
- The null literal \perp may only be used in equality terms ($=, \neq$). \diamond

We use an expression evaluation function to evaluate a linear boolean expression to true or false given a variable assignment.

Definition 2.9 (Expression evaluation function). Let U be the universe of all variable values. Let $V_P \subseteq V$ be a set of process variables with domain function $\text{dom} : V_P \rightarrow \mathbb{P}(U)$. Let $\text{expr} \in \text{EXPR}(V_P)$ be a boolean expression. We evaluate the truth value of a boolean expression with an *evaluation function*:

$$\text{eval}_{V_P, \text{dom}} : (\text{EXPR}(V_P) \times \mathcal{U}_{\text{dom}}^P) \rightarrow \{\text{true}, \text{false}\}.$$

Function $\text{eval}_{V_P, \text{dom}}$ evaluates the expression to either true or false given a variable assignment $w \in \mathcal{U}_{\text{dom}}^P$. If one or more variables required to evaluate the expression are undefined, i. e., $v \notin \text{dom}(w)$, then false is returned. \diamond

We do not elaborate further on the implementation of the evaluation function since it is based only on basic arithmetic and basic logic.

Example 2.3 (Linear boolean expression). Given variables $a, b, c \in V_P$ that are defined over domains $\text{dom}(a) = \{\text{true}, \text{false}\}$, $\text{dom}(b) = \mathbb{R}$, and $\text{dom}(c) = \mathbb{N}$, we can define a boolean expressions: $\text{expr} \leftarrow ((a = \text{true}) \vee ((\frac{2}{3} \cdot a) < b))$. We evaluate the expression using an evaluation function that follows the standard rules for mathematical terms and boolean expressions. The expression is fulfilled, e. g., for the variables values $a = \text{true}$, $a = 3$, and $b = 2$ and for the variables values $a = \text{false}$, $a = 3$, and $b = 3$. The expression is not fulfilled for the variable assignment $a = \text{false}$, $a = 3$, and $b = 2$.

2.3 Event Logs

An event log stores data about the occurrence of activities that were recorded by information systems while supporting the execution of a process. Each execution of a *process instance* results in a sequence of events [Aal16].

Definition 2.10 (Event log [Aal16]). Let U be an universe of values. Let $V_L \subseteq V$ be a finite set of observed variables, which we denote as *attributes*. An event log L is a 4-tuple $L = (E, \Sigma, \#, \mathcal{E})$, in which:

- E is a non-empty, finite set of unique event identifiers,
- $\Sigma \subseteq U$ is a non-empty, finite set of activity names,
- $\# : E \rightarrow \mathcal{U}_{\text{dom}}^L$ retrieves the attribute values assigned to an event, and
- $\mathcal{E} \subseteq E^*$ is the finite set of traces over E .

The same event may not appear twice in a trace, i. e., $\forall_{(e_1, \dots, e_n) \in \mathcal{E}} \forall_{1 \leq i \leq n} \forall_{1 \leq j \leq n} (i \neq j \implies e_i \neq e_j)$; or in two different traces, i. e., $\forall_{e \in \mathcal{E}} \forall_{\sigma_1 \in \mathcal{E}} \forall_{\sigma_2 \in \mathcal{E}} ((e \in \sigma_1 \wedge e \in \sigma_2) \implies \sigma_1 = \sigma_2)$. \diamond

A trace $\sigma \in \mathcal{E}$ records the sequence of events for *one process instance*. Each event represents the execution of an activity in the process. Given an event $e \in E$, we write

$\#_a(e) \in U$ to obtain the value $u \in U$ recorded for attribute $a \in V_L$. One mandatory attribute is recorded by each event: $\#_{\text{activity}}(e) \in \Sigma$, the **name of the activity** that caused the event. An event log can be seen as an ordered table of values, which may be sparse. Each row represent one unique event and each column an attribute. Some columns of the table are used to group events into traces and uniquely identify events. These columns allow viewing the event log as a set of ordered traces. We define three more operations on event logs.

Definition 2.11 (Operations on event logs). Let $L = (E, \Sigma, \#, \mathcal{E})$ be an event log. Let $\sigma = \langle e_1, \dots, e_n \rangle \in \mathcal{E}$ be a trace from L .

- The *latest attribute values* before an event occurred are $\text{latest} : E \rightarrow \mathcal{U}_{dom}^L$, i. e., $\text{latest}(e_1) = \emptyset$ and $\text{latest}(e_i) = \text{latest}(e_{i-1}) \oplus \#(e_{i-1})$.
- The *predecessor event* of e_i in trace σ is returned by $\bullet e_i \in (E \cup \{\perp\})$, i. e., $\bullet e_1 = \perp$ and $\bullet e_i = e_{i-1}$.
- The *successor event* of e_i in trace σ is returned by $e_i \bullet \in (E \cup \{\perp\})$, i. e., $e_n \bullet = \perp$ and $e_i \bullet = e_{i+1}$. \diamond

Example 2.4 (Event log). Table 2.1 shows four traces $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ from an event log that was recorded by information systems supporting the hospital process introduced in Example 1.1. Each event has a unique identifier that can be used to access the data stored in the attributes, e. g., we can identify the activity of event e_{10} as $\#_{\text{activity}}(e_{10}) = \text{Triage}$.

Contextual data about the executed activities is stored as further attributes, e. g., event e_{10} writes the timestamps of the activity execution, $\#_{\text{time}}(e_{11}) = 23/11/16 12:00$, the resource and the organizational role executing the activity, $\#_{\text{role}}(e_{10}) = \text{Nurse}$ and $\#_{\text{resource}}(e_{10}) = \text{Terry}$, and the triage color assigned to the patient: $\#_{\text{time}}(e_{11}) = \text{Yellow}$. Some events record more attributes than others. For example, event e_{17} records the attribute $\#_{\text{referral}}(e_{17}) = \text{Ward}$ indicating the kind of transfer. Event e_{28} does not record a resource, i. e., resource $\notin \text{dom}(\#(e_{28}))$.

We access the latest attribute values recorded before e_{18} occurred as $\text{latest}(e_{18}) = (\text{activity} \mapsto \text{Decide}, \text{time} \mapsto 23/11/16 15:30, \text{role} \mapsto \text{Doctor}, \text{resource} \mapsto \text{Danny}, \text{color} \mapsto \text{Yellow}, \text{referral} \mapsto \text{Ward})$. Function *latest* collects all attribute values that are recorded in the trace. Older values are overwritten by newer values, i. e., only the latest attribute values are retained.

In some cases, we use a simplified view on event logs. A *simplified event log* stores only minimal data about the process. All contextual data, i. e., all attributes except the activity name and the sequence order, are omitted. It is defined as a multiset of sequences over the set of activity names [Aal16]: $SL \in \mathbb{B}(\Sigma^*)$. Figure 2.1 depicts

Table 2.1: Four traces of an event log recorded for the hospital process.(a) Trace σ_1 , a patient that is transferred to the intensive care unit.

id	activity	time	role	resource	color	referral
e ₁₀	Triage	23/11/16 12:00	Nurse	Terry	Yellow	
e ₁₁	Register	23/11/16 12:20	Nurse	Nancy		
e ₁₂	Check	23/11/16 12:50	Nurse	Nancy		
e ₁₃	Check	23/11/16 13:40	Nurse	Nancy		
e ₁₄	Check	23/11/16 14:35	Nurse	Nancy		
e ₁₅	Visit	23/11/16 14:40	Doctor	Danny		
e ₁₆	Diagnostic	23/11/16 15:10	Specialist	Ray		
e ₁₇	Decide	23/11/16 15:30	Doctor	Danny		Ward
e ₁₈	Prepare	23/11/16 15:35	Nurse	Nancy		
e ₁₉	Transfer	23/11/16 16:00	Nurse	Nathan		

(b) Trace σ_2 , a patient that is transferred to another hospital.

id	activity	time	role	resource	color	referral
e ₂₀	Triage	5/12/16 21:00	Nurse	Terry	Red	
e ₂₁	Register	5/12/16 21:02	Nurse	Nathan		
e ₂₂	Check	5/12/16 21:15	Nurse	Nathan		
e ₂₃	Diagnostic	5/12/16 21:35	Specialist	Ray		
e ₂₄	Visit	5/12/16 21:50	Doctor	Danny		
e ₂₅	Check	5/12/16 22:25	Nurse	Nathan		
e ₂₆	Decide	5/12/16 22:30	Doctor	Danny		
e ₂₇	Prepare	5/12/16 22:32	Nurse	Nathan		Tertiary
e ₂₈	Organize Ambulance	5/12/16 23:15	System			
e ₂₉	Discharge	5/12/16 23:16	Nurse	Nathan		

(c) Trace σ_3 , a patient that is not admitted to the emergency ward.

id	activity	time	role	resource	color	referral
e ₃₀	Triage	12/12/16 10:30	Nurse	Terry	White	
e ₃₁	Register	12/12/16 10:45	Nurse	Nathan		

(d) Trace σ_4 , a patient that is not admitted to the emergency despite its triage color.

id	activity	time	role	resource	color	referral
e ₄₀	Triage	15/12/16 10:30	Nurse	Terry	Green	
e ₄₁	Register	15/12/16 10:45	Nurse	Nathan		
e ₄₂	Check	15/12/16 11:15	Nurse	Nathan		

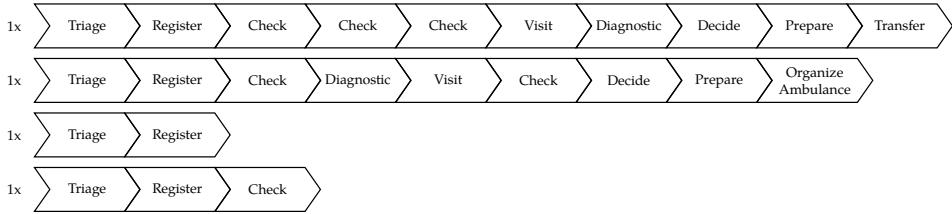


Figure 2.1: A simple event log obtained by transforming the four traces of the example event log shown in Table 2.1.

a visual representation of a simplified event log based on the four example traces $\sigma_1, \sigma_2, \sigma_3, \sigma_4$. Each event is depicted by a chevron shape that is labeled with the activity name. A trace is a sequence of chevron symbols. Its multiplicity is written in front of the chevron sequence. In the remainder of this thesis, we use this simplified view for a compact representation visual representation of event logs for which only the control-flow perspective is important.

2.4 Decision Trees

A decision tree is a visual representation of a set of disjoint decision rules that can be used as classifier. Each decision rule predicts a target class based on an observed set of attribute values. In a decision tree, each internal node (i. e., nodes with at least one child node) specifies a test concerning a single attribute. Each branch from an internal node represents a possible outcome of that test, e. g., a test on a Boolean attribute X yields the branches $X = \text{true}$, $X = \text{false}$. Leaf nodes represent the target class after a series of tests (i. e., the final prediction).

Decision trees can be learned based on observation instances (also denoted as training instances), e. g., by using C4.5 [Qui93].

Definition 2.12 (Observation Instances). Let C be a set of target classes. Let $V_L \subseteq V$ be a set of observed variables and let U be a universe of possible values. We denote with

$$OI \in \mathbb{B}(U_{dom}^L \times C)$$

a multi-set of observation instances. An *observation instance* $(w, c) \in OI$ records that target class c was observed for attribute values w . \diamond

We define a *decision tree builder function* that returns a set of guard expressions based on a decision tree learned on observation instances. Each guard expression returned by the decision tree builder function predicts a target class. Here, we do not elaborate on how the observation instances are extracted from a given event log since this is described later as part of our method in Chapter 10.

Definition 2.13 (C4.5 Decision Tree Builder). Let C be a set of target classes. Let $OI \in \mathbb{B}(\mathcal{U}_{dom}^L \times C)$ be a multi-set of observation instances over a set V_L of observed event log variables with values U . Let EXPR_{V_L} be the universe of boolean expressions over the set of observed variables. Let $mi \in \mathbb{N}$ be the minimum number of instances on a leaf for the splitting criterion in the decision tree induction. We denote with

$$\text{buildTree}_{C,mi}(OI) \in \mathbb{P}(\text{EXPR}(V_L) \times C)$$

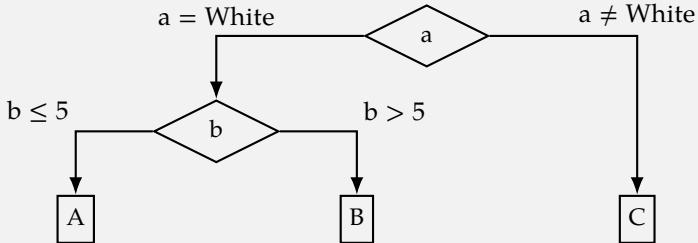
a function that returns the leafs of a decision tree trained with C4.5 [Qui93] using the instances OI . A *leaf* $(expr, c) \in \text{buildTree}_{C,mi}(OI)$ predicts *target class* $c \in C$ under *condition* $expr \in \text{EXPR}(V_L)$. \diamond

The rule for a leaf of the decision tree is obtained by taking the conjunction of all conditions represented by those nodes that are encountered on a path from the leaf up to the root node [LA13b].

Example 2.5 (Decision Tree Builder). Assume target classes $C = \{A, B, C\}$ and the following multi-set of 40 observation instances:

$$\begin{aligned} OI = [& ((a \mapsto \text{White}, b \mapsto 10), A)^{10}, \\ & ((a \mapsto \text{White}, b \mapsto 5), B)^{10}, \\ & ((a \mapsto \text{Green}, b \mapsto 10), C)^{10}, \\ & ((a \mapsto \text{Red}, b \mapsto 10), C)^{10}] \end{aligned}$$

C4.5 would learn the following decision tree when trained on the observation instances OI :



Based on such a decision tree, we obtain the set of leafs as:

$$\begin{aligned} \text{buildTree}_{C,mi}(OI) = & \{(b \leq 5 \wedge a = \text{White}, A), \\ & (b > 5 \wedge a \neq \text{White}, B), \\ & (a \neq \text{White}, C)\} \end{aligned}$$

3 Process Models

Process models describe the behavior of processes. Many different process modeling notations have been proposed [Aal16]. In this chapter, we introduce three notations that are used in the remainder of this thesis. Each notation has different properties, which makes it applicable in a certain setting. We position the notations used in this thesis with regard to other notations and show that results can be transferred to standard notations used in practice (e.g., to BPMN).

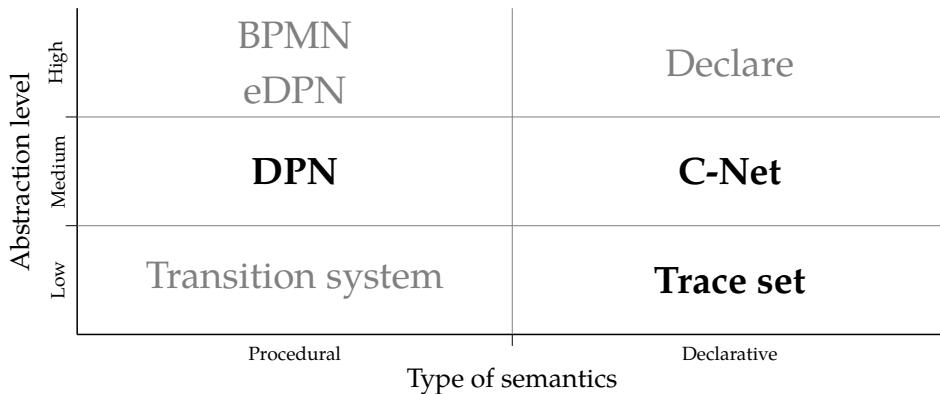


Figure 3.1: Process model notations used in this thesis compared to other notations. The notations are categorized based on their level of abstraction and their kind of semantics.

Figure 3.1 provides an overview of the employed process modeling notations. We use three different process model notations to unambiguously specify the expected behavior of a process:

1. **Trace set** (Section 3.1),
2. **Data Petri Nets (DPNs)** (Section 3.2), and
3. **Causal Nets (C-Nets)** (Section 3.3).

The **trace set notation** enumerates the set of possible process executions. We use this notation whenever the method does not depend on the actual process modeling notation. Consequently, methods presented in this manner are independent of the particular formalism (e.g., Petri nets [Pet62; Rei85], EPC [KSN92], BPEL [BPEL07], UML activity diagrams, Declare [PSA07], BPMN) employed to model processes.

In practice, higher level notations are needed to describe non-trivial processes, e. g., to describe processes with an infinite set of possible execution traces. Therefore, we use two notations with a graphical notation to describe the implementation of the methods presented in this thesis. Primarily, we use the **DPN notation**, which has clear semantics and is based on Petri nets, which have a strong formal foundation. Therefore, we use DPNs whenever a process model needs to be specified as input to a presented method, i. e., in Chapters 5, 6, 9 and 10.

For the presentation of the process discovery method in Chapter 8, we employ the **C-Net notation**. C-Nets have declarative semantics and are tailored towards process discovery [AAD11]. They are a better match for this particular problem than DPNs.

Finally, we show in Section 3.4 that both C-Nets and DPNs can be transformed to languages on a higher abstraction level. Thus, the methods presented in this thesis can be generalized towards process modeling languages that are used in practice. We selected BPMN as an example for the de-facto standard process modeling notation in practice and extended Data Petri Nets (eDPN) [Bal16] as an example for a high-level notation that is based directly on the DPN semantics.

3.1 Process Behavior Expressed as a Trace Set

A very basic notation of a process model is the enumeration of all possible process executions. Each process execution (i. e., case) is a sequence of *process steps*. Each process step represents an activity execution together with recorded variable values.

Definition 3.1 (Process step). Given universes of values U , process variables $V_P \subseteq V$, and process transitions T , we denote with $PS = T \times \mathcal{U}_{dom}^P$ the *set of all possible process steps*. A process step $(t, w) \in PS$ describes the execution of a process transition t together with the variable assignment \mathcal{U}_{dom}^P . \diamond

The set of all possible sequences of process steps defines the behavior of the process. This definition of a process model based on its valid traces abstracts from the modeling notation. Any notation with executable semantics can be seen in this manner. We denote the set of execution traces as a *Trace Set*.

Definition 3.2 (Trace set). Given universes of values U , process variables $V_P \subseteq V$, and process transitions T , and let $PS = T \times \mathcal{U}_{dom}^P$ be the *set of all possible process steps*. A *process trace* is a finite sequence of process steps $\sigma \in PS^*$, which corresponds to one valid execution (i. e., process instance) of the process. Then, a *trace set* $TS \subseteq PS^*$ is a (possibly infinite) set of process traces. \diamond

Note that, Definition 3.2 does not provide any kind of abstraction from the low-level behavior of the process, e. g., repeating activities lead of an infinite set of possible process executions.

Example 3.1 (Trace set of the hospital process). We can describe the hospital process (cf., Example 1.1) with a trace set. The set of traces is infinite since that process includes a loop: "While patients are in the emergency ward the nurse periodically checks their condition". Therefore, we show only an exemplary process execution. The set of process transitions is: $T = \{t_{tri}, t_{reg}, t_{che}, t_{dia}, t_{vis}, t_{dec}, t_{pre}, t_{org}, t_{vis}, t_{obs}, t_{dis}\}$. The set of process variables is: $V_P = \{\text{data}_{color}, \text{data}_{referral}, \text{time}_{che}, \text{res}_{reg}, \text{res}_{pre}\}$. Each process transition models an activity from Example 1.1, e.g., t_{tri} corresponds to the *Triage* activity and t_{dia} corresponds to taking a medical *Diagnostic Test*. Each variable captures a multi-perspective aspect of the process, e.g., data_{color} records the triage color; time_{che} records the last execution time of the respective process transitions t_{reg} , t_{che} , and t_{pre} ; and variable res_{reg} stores the name of the nurse carrying out the activities *Register*. A possible process trace is:

$$\begin{aligned} & ((t_{tri}, (\text{data}_{color} \mapsto \text{Yellow})), (t_{reg}, (\text{time}_{che} \mapsto 23/11/16 12:20, \text{res}_{reg} \mapsto \text{Nancy})), \\ & (t_{che}, (\text{time}_{che} \mapsto 23/11/16 12:50)), (t_{che}, (\text{time}_{che} \mapsto 23/11/16 13:20)), \\ & (t_{che}, (\text{time}_{che} \mapsto 23/11/16 13:50)), (t_{vis}, \emptyset), \\ & (t_{dia}, \emptyset), (t_{dec}, (\text{data}_{referral} \mapsto \text{ICU})), \\ & (t_{pre}, (\text{time}_{che} \mapsto 23/11/16 14:20, \text{res}_{pre} \mapsto \text{Nancy}), (t_{tra}, \emptyset)) \in TM. \end{aligned}$$

This process trace fulfills all multi-perspective constraints that are described in Example 1.1. For example, the triage color of the admitted patient is different from *white*, the activity *Check* is done every hour until activity *Prepare* occurs, and the same nurse registers and prepares the patient for discharge.

Process steps correspond to executions of activities and assignments of attribute values to process variables (e.g., as recorded attribute assignments in an event log). It is possible that the same activity is represented by two distinct process transitions. Similarly, it is possible that the same event log attribute is represented by two distinct process variables. This duplication can be used to distinguish, e.g., the first execution of an activity from its second execution and the timestamp of the first execution of an activity from the timestamp of its second execution. Therefore, we introduce two labeling functions connecting process transitions to activities and process variables to attributes.

Definition 3.3 (Labeled trace set). Let U be an universe of values. Let $V_P \subseteq V$ be a set of process variables and let $V_L \subseteq V$ be the set of attributes of an event log such that $V_P \cap V_L = \emptyset$. Let T be a set of process transitions and let $\Sigma \subseteq U$ be the set of activity names. A *labeled trace set* is a triple (TS, λ, ν) where:

- $TS \subseteq PS^*$ is a trace set,
- $\lambda : T \rightarrow (\Sigma \cup \{\tau\})$ is an activity label function that returns the observable activity name of a process transition or τ in case of unobservable internal process transitions,

- $v : V_P \rightarrow V_L$ is a variable label function that returns the observable attribute name of a process variable. \diamond

Including unobservable routing transitions in Definition 3.3 may seem not to be strictly necessary. However, we use them later when defining the language of a DPN (cf., Definition 3.12) in terms of process traces.

3.2 Data Petri Nets

We adopt *DPNs* [LA13a; LA13b; SST11] as language for modeling multi-perspective processes that are used as *input* to the developed methods. DPNs are based on Petri nets, which we introduce in the next section.

3.2.1 Petri Nets

Petri nets are used to model the control-flow perspective of a process. Only the ordering of activity executions is described and all other perspectives on a process are ignored. A Petri net is a bipartite directed graph that consists of places and transitions. Transitions represent the activities of a process and places capture the current state of the process. Transitions are depicted as rectangles and places are depicted as circles.

Definition 3.4 (Petri net). A Petri net is a triple (P, T, F) with:

- P is a finite set of places,
- T is a finite set of transitions, and
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of flow relations that describe a bipartite graph between places and transitions. \diamond

Places can be *marked* with zero or more *tokens*. We denote with $\bullet t = \{p \in P \mid (p, t) \in F\}$, the *input places* of a transition t ; and with $t^\bullet = \{p \in P \mid (t, p) \in F\}$, the *output places* of a transition t . Definitions of pre- ($\bullet p$) and postsets ($p\bullet$) of places are analogous.

The distribution of tokens over all places of a Petri net determines its state. Each place may be associated with zero or more tokens. The set of possible *states* of the Petri net is formed by the multiset of its places $\mathbb{B}(P)$. A state $M \in \mathbb{B}(P)$ is called **marking** of the Petri net. We introduce two special kind of markings to define the complete behavior of a Petri net:

- the *initial marking* $M_I \in \mathbb{B}(P)$ describing the start state, and
- the *final marking* $M_F \in \mathbb{B}(P)$ indicating the end state.

Executions of the Petri net start with the initial marking and end in the final marking. The marking of a Petri net is changed by firing transitions (i. e. executing process

activities). Transitions in a Petri net may be *fired* only when they are *enabled*. Transitions are enabled only if each of their input places is marked with one or more tokens. Firing a transition in a Petri net *consumes* one token from each of its input places and *produces* one token to each of its output places. This possibly enables other transitions and changes the marking. Executions of a Petri net finish when the final marking M_F is reached and none of the transitions is enabled.

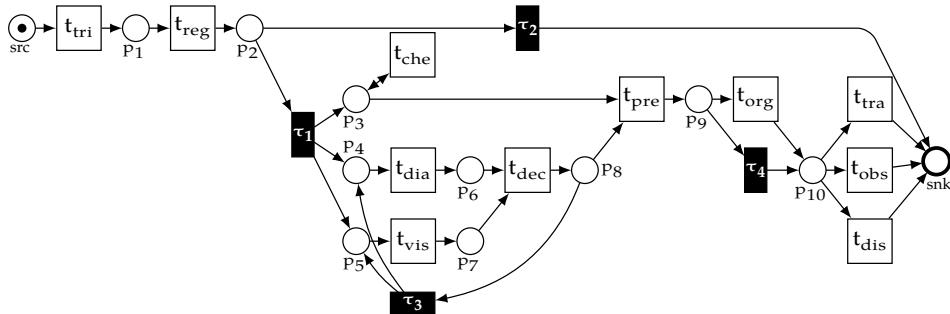


Figure 3.2: A Petri net modeling the control-flow perspective of the hospital process.

Example 3.2 (Petri net). Figure 3.2 shows a Petri net modeling the hospital process that we introduced by a natural language description in Example 1.1. Each valid execution sequence (i. e., process trace) of the Petri net corresponds to a process instance (i. e., case) of the hospital process. The initial marking consists of one token that is put in place *src*. Tokens are depicted as small black dots within places. We abbreviate the identifiers of the transitions by using the first three letters of the name of the represented activity. Please note that not all transitions refer to visible process activities (i. e., some may be routing transitions) and that two transitions may refer to the same activity (i. e. duplicate transitions). Routing transitions are also known as τ -transitions or invisible transitions. Later, in Definition 3.12, they are introduced formally.

In the initial state, only the transition t_{tri} (Triage) is enabled. Upon executing transition t_{tri} one token is consumed from place *src* and a new token is produced in place *p₁*. The marking of the Petri net changes from the initial marking $M_I = [src]$ to $[p_1]$. With the new marking, transition t_{ref} (Register) is enabled since its input place *p₁* is marked with one token. Executing transition t_{tri} consumes one token from *p₁* and produces one token in *p₂*. Now, both τ_1 and τ_2 are enabled in marking $[p_2]$. An *exclusive choice (XOR split)* is modeled between the two routing transitions. Executing τ_1 removes the token from *p₂* and, in turn, disables transition τ_2 . Assume τ_1 has been executed, then, the resulting marking is $[p_3, p_4, p_5]$. Transition τ_1 splits the thread of control in three *concurrent branches*.

(*AND split*). Therefore, transitions t_{che} (Check), t_{dia} (Diagnostic), and t_{vis} (Visit) are enabled. Firing one of these transitions does not disable the other, e. g., both t_{che} and t_{dia} can be executed independently from each other. Transition t_{che} can be repeated since it consumes a token from p_3 and put new token back in the same place p_3 . Transitions t_{dec} (Decide) *merges two of the concurrent branches back together (AND join)*. The transition consumes tokens from both p_6 and p_7 but produces only one token in place p_9 . Afterwards, either transition t_{pre} (Prepare) may be executed or both transitions t_{dia} and t_{vis} may be executed again. If transition t_{pre} is executed, two tokens are consumed: one token from place p_3 and one token from place p_8 . Thus, the two remaining concurrent branches are joined together. Then, there are two more exclusive choices modeled regarding transition t_{org} (Organize Ambulance), which can be skipped; and transitions t_{tra} (Transfer), t_{dis} (Discharge), and t_{obs} (Observe), which are in conflict with each other. Finally, one token is put in place snk . The final marking is $[\text{snk}]$ and no further transition in the Petri net is enabled, therefore, the process instance ends.

We only consider Petri nets with a single source place in the initial marking and a single sink place in the final marking. Moreover, we restrict transitions to only remove and produce one token on each input- and output arc (i. e. arcs are unweighted). This is similar to the definition of workflow nets [Aal98] and avoids introducing unnecessary notation. We omitted a formal introduction of the Petri net execution semantics, since we extend them to the more expressive DPN. A comprehensive introduction on classical Petri nets is given, e. g., by Murata [Mur89].

3.2.2 Syntax and Semantics of DPNs

A DPN is a Petri net, in which transitions can manipulate variables and are associated with a data-dependent guard expressions (guards). Additional perspectives of the process are captured by storing contextual information in variables and expressing constraints with guards. For example, we can store information on the organizational resources, which executed activities, in variables over a nominal domain. Moreover, we can express resource-perspective constraints such as the four-eyes principle (two activities should not be executed by the same resource) as guards, e. g., $\text{resource}_A \neq \text{resource}_B$. Guards are boolean expressions over the variables of the DPN, i. e. guards evaluate to either true or false.

Differently from notations such as Colored Petri nets (CPN) [JK09], which store data values locally using colored tokens, the variables in a DPN are globally defined and data values in a DPN are visible to all transitions. This is useful in the process mining setting that we consider in this thesis since we do not know the visibility of data values in the processes under investigation.

Definition 3.5 (Prime Variables). Let U be the universe of variable values. Let $V_P \subseteq V$ be a set of process variables. We denote with $V'_P = \{v' \mid v \in V_P\}$ a set of additional *prime variables*. One prime variable is paired with each variable $v \in V_P$. \diamond

We use a prime variable $v' \in V'_P$ to distinguish the new data value that is written by a transition to variable $v \in V_P$ from its old value before the transition is executed.

Definition 3.6 (Data Petri Net [LA13a]). Let U be the universe of variable values. Let $V_P \subseteq V$ be a set of process variables and let V'_P be a set of prime variables. A DPN $N = (P, T, F, V_P, dom, in, wr, gd)$ is a Petri net with additional components, which can be used to describe the additional perspectives of the process model:

- (P, T, F) is a Petri net,
- V_P is a finite set of process variables,
- $dom : V_P \rightarrow \mathbb{P}(U)$ returns the (potentially) infinite domain for variables $v \in V_P$,
- $in : V_P \rightarrow U$ returns the initial value for variables $v \in V_P$,
- $wr : T \rightarrow \mathbb{P}(V_P)$ returns the set of required *write operations* for transitions $t \in T$,
- $gd : T \rightarrow \text{EXPR}(V_P \cup V'_P)$ returns the boolean guard expression that is associated with each transition $t \in T$. The guard is defined over variables V_P and the additional *prime variables* V'_P .⁵ \diamond

Variables in a DPN are updated by transitions through *write operations*. We assume that variables are assigned an initial value as specified by function *in*. In the remainder, we use the symbol \perp as a placeholder for the initial value of any variable. Guard expressions are associated with transitions. Guard expressions in short guards further constrain when transitions may be executed. A transition t in a DPN can be executed only if all its *input places contain at least one token* and its *guard is satisfied*. The guard is evaluated based on the values written for regular variables ($v \in V_P$) before the execution of transition t and the values written for prime variables ($v' \in V'_P$) by transition t itself.

Example 3.3 (DPN). Figure 3.3 shows a DPN modeling the hospital process that we introduced by a natural language description in Example 1.1. Each valid execution sequence (i.e., process trace) of the DPN corresponds to a process instance (i.e., case) of the hospital process. The underlying Petri net structure of the process is modeled in the same way as in Figure 3.2. The Petri net models the control-flow perspective of the process. Other perspectives (e.g., resources, time, and data) are expressed with the additional modeling elements of a DPN: variables, write functions, and guard expressions. Variables are depicted with yellow colored hexagons. In Figure 3.3, we added variables $\text{data}_{\text{color}}$ and $\text{data}_{\text{referral}}$ to

⁵If a transition t is associated with no guard, we set $gd(t) = \text{true}$.

capture the data perspective; variable time_{che} to capture the time perspective; and variables res_{reg} and res_{pre} to capture the resource perspective. Variables are written by transitions, e.g., the *dashed edge* from transition t_{tri} to variable $\text{data}_{\text{color}}$ denotes that a value needs to be assigned to variable $\text{data}_{\text{color}}$ when executing transition t_{tri} . Variable $\text{data}_{\text{color}}$ is part of the set of *write operations* $wr(t_{\text{tri}})$. The values of variables are globally visible to all transitions that use the current values to evaluate guard expressions. Guard expression add additional constraints to the execution of a transition. Transitions that are assigned a non-trivially fulfilled guard (i.e., $gd(t) \neq \text{true}$) are drawn with a double border. For example, transition τ_2 can only be executed when the corresponding expression ($\text{data}_{\text{color}} = \text{White}$) is fulfilled. Thus, only patients with the triage color *white* may directly leave the emergency ward. Similarly, constraints are also specified on the transitions t_{org} , t_{tra} , t_{obs} , and t_{dis} . Their execution depends on the decision regarding the transfer of the patient, which is written to variable $\text{data}_{\text{referral}}$ by transition t_{dec} .

Variables and guards can also be used to model constraints on the time perspective of the process. For example, transitions t_{reg} , t_{che} , and t_{pre} record their execution time with variable time_{che} . Using this variable, we can implement the constraint that patients need to be checked at least every hour after they have been registered until they are prepared for discharge or transfer. The guard of t_{che} specifies the last execution time recorded time_{che} and current execution time recorded $\text{time}'_{\text{che}}$ should be at most 1 hour apart from each other. A similar guard is added to transition t_{pre} . Boths guard reference a special prime variable $\text{time}'_{\text{che}}$ to access the variable value written by the current transition firing.

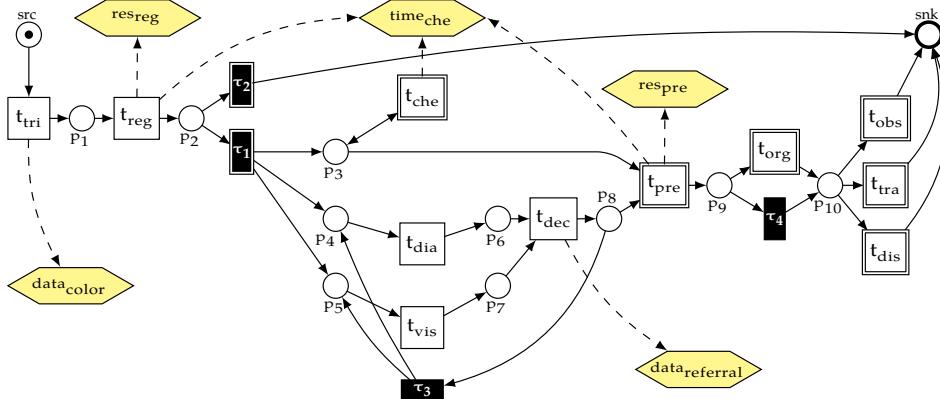
The resource perspective of the process is also specified by variables and guards. For example, we implement the *retain familiar* regarding transitions t_{reg} and t_{pre} (i.e., the same nurse shall carry out both activities) using two variables res_{reg} and res_{pre} and add the condition $\text{res}_{\text{reg}} = \text{reg}'_{\text{pre}}$ to the guard expression of transition t_{pre} . Both variables store the resource that executed the transition. Transition t_{pre} may only be fired if both resources match.

We now introduce the execution semantics of a DPN stepwise by defining (1) its state, (2) the set of valid transition firings of a DPN, and (3) the transition firing rule, which changes the state of the DPN. The state of a DPN is the combination of both the marking of the Petri net and the current variable assignment.

Definition 3.7 (State of a DPN [LA13a]). Let $N = (P, T, F, V_P, dom, in, wr, gd)$ be a DPN. The set of possible *states* of N is formed by all pairs (M, α) where:

- $M \in \mathbb{B}(P)$, i.e., is the marking of the Petri net (P, T, F) , and
- $\alpha \in U_{dom}^P$ is the current variable assignment.

◇



(a) Written variables are depicted as dashed edges to yellow hexagons. Guarded transitions are drawn with a double border.

Transition	Guard expression
τ_1	$\text{data}_{\text{color}} \neq \text{White}$
τ_2	$\text{data}_{\text{color}} = \text{White}$
t_{che}	$\text{time}'_{\text{che}} \leq (\text{time}_{\text{che}} + 1\text{h})$
t_{pre}	$\text{time}'_{\text{che}} \leq (\text{time}_{\text{che}} + 1\text{h}) \wedge \text{res}_{\text{reg}} = \text{res}'_{\text{pre}}$
t_{org}	$\text{data}_{\text{referral}} = \text{Tertiary}$
t_{obs}	$\text{data}_{\text{referral}} \neq \text{Home}$
t_{tra}	$\text{data}_{\text{referral}} \neq \text{Home}$
t_{dis}	$\text{data}_{\text{referral}} = \text{Home}$

(b) Guards assigned to transitions that are not always enabled (i.e., $gd(t) \neq \text{true}$).

Figure 3.3: Multiple perspectives of the hospital process modeled with a DPN.

We denote with α_I a special *initial variable assignment* in which every variable is assigned its initial value: $\forall_{v \in V_P} (\alpha_0(v) = in(v))$.

In any state, zero or more transitions of a DPN-net may be able to fire. Firing a transition $t \in T$ moves tokens from the input places $\bullet t$ to the output places $t \bullet$ and writes values to the variables defined by the write operations $wr(t)$. Transition can only fire if their guard ($gd(t)$) is fulfilled according to the current and written variable values.

Definition 3.8 (Valid transition firing). Let U be the universe of variable values. Let $N = (P, T, F, V_P, dom, in, wr, gd)$ be a DPN-net. Let $eval_{V_P, dom}$ be an expression evaluation function. Let $PS : T \times U_{dom}^P$ be the set of possible process steps (cf. Definition 3.2). A *transition firing* s in a DPN is a process step $(t, w) \in PS$. Transition firing $s = (t, w)$ is **valid** in a state (M, α) of the DPN if four conditions are satisfied:

1. each input place of t contains at least one token, i. e., $\forall_{p \in \bullet t} (M(p) > 0)$;
2. the transition t writes exactly the variables specified by the write operation function, i. e., $dom(w) = wr(t)$;
3. the value written to each variable is valid according to its domain, i. e., $\forall_{v \in dom(w)} (w(v) \in dom(v))$; and
4. the guard $gd(t)$ is fulfilled when evaluated on variable assignment $\alpha' \in U_{dom}^P$, i. e., $eval_{V_P \cup V'_P, dom}(gd(t), \alpha') = \text{true}$. Variable assignment α' consists of the current variable assignment (α) for normal variables V_P and the assignment to variables that are to be written (w) for prime variables V'_P :

$$\forall_{v \in V_P \cup \{v' \in V'_P | v \in w(t)\}} \left(\alpha'(v) = \begin{cases} \alpha(v) & \text{if } v \in V_P \\ w(v) & \text{otherwise.} \end{cases} \right)$$

All other transition firings are *invalid* in state (M, α) . \diamond

Example 3.4 (Transition firing in a DPN [LA13a]). Assume that the DPN specified in Figure 3.3 is in the state $([\text{src}], \alpha_1)$, i. e., there is a token in the initial place and all variables are assigned their initial values. Transition t_{tri} is the only transition that can be fired in this state since only place src is marked with a token. The transition is not guarded (i. e., $gd(t_{\text{tri}}) = \text{true}$). However, the transition writes the variable $\text{data}_{\text{color}}$. Therefore, a valid firing needs to assign a value to variable $\text{data}_{\text{color}}$. For example, possible valid transition firing would be $(t_{\text{tri}}, (\text{data}_{\text{color}} \mapsto \text{White}))$.

Another example, assume that the DPN is in the state (M_1, α_1) with $M_1 = [p_2]$ and $\alpha_1 = \alpha_I \oplus (v_{\text{color}} \mapsto \text{White})$ (i. e., only variable v_{color} is assigned a value). According to the standard Petri net execution semantics both transitions τ_1 and τ_2 would be enabled. However, since both transition are assigned guards, their guard needs to be fulfilled in order to fire. In this case, the only valid transition firing is (τ_2, \emptyset) , i. e., the patients needs to leave the emergency ward.

When firing a valid transition the state of the DPN changes from state (M, α) to the next state (M', α') . Firing a single valid transition can be extended to sequences of valid transition firings, i. e., process traces $\sigma \in PS^*$.

Definition 3.9 (Process trace of a DPN [LA13a]). Let $N = (P, T, F, V_P, dom, in, wr, gd)$ be a DPN-net and (M, α) be a state of the DPN. Let $s = (t, w)$ be a valid transition firing in (M, α) . Firing s in state (M, α) leads to state (M', α') . In the new state the new marking is calculated according to the Petri net execution semantics and the variable assignment is updated based on the write operations:

$$\bullet \quad \forall_{p \in P} M'(p) = \begin{cases} M(p) - 1 & \text{if } p \in \bullet t, \\ M(p) + 1 & \text{if } p \in t \bullet, \\ M(p) & \text{otherwise;} \end{cases}$$

$$\bullet \quad \forall_{v \in V} \alpha'(v) = \begin{cases} \alpha(v) & \text{if } s \notin dom(w), \\ w(v) & \text{otherwise.} \end{cases}$$

This is denoted as $(M, \alpha) \xrightarrow{s} (M', \alpha')$. By firing multiple transitions we obtain a process trace $\sigma_N = \langle s_1, \dots, s_n \rangle \in PS^*$, i. e., $(M_0, \alpha_0) \xrightarrow{\sigma_N} (M_n, \alpha_n)$ corresponds to:

$$(M_0, \alpha_0) \xrightarrow{s_1} (M_1, \alpha_1) \xrightarrow{s_2} \dots \xrightarrow{s_n} (M_n, \alpha_n).$$

◊

(M_I, α_I) is the **initial state** of a DPN, i. e., the initial marking of the Petri net and each variable mapped to its initial value. We define the set of possible process traces as those traces that start in the initial state of the DPN and lead to some other state of the DPN, i. e., not necessarily to a final state.

Definition 3.10 (Possible process traces). Let $N = (P, T, F, V_P, dom, in, wr, gd)$ be a DPN-net. Let $M_I \in \mathbb{B}(P)$ be the initial marking. We denote the set of possible process traces that start in the initial state (M_I, α_I) and lead to any state (M, α) as:

$$TS_{N, M_I} = \{\sigma_N \in PS^* \mid \exists_{M \in \mathbb{B}(P)} \exists_{\alpha \in U_{dom}^P} ((M_I, \alpha_I) \xrightarrow{\sigma_N} (M, \alpha))\}.$$

◊

The overall trace set (i. e., its entire behavior) of a DPN corresponds to all possible process traces starting from an initial state and ending in any final state. The set of **final states** includes every state (M_F, α) , i. e., all states in which the final marking of the Petri net is reached regardless of the variable values. Thus, the variable assignment reached in a final state does not matter for the proper completion of an execution. Still, it is possible to enforce a certain variable assignment in all final state; guards can be assigned to the transitions leading to a final marking.

Definition 3.11 (Trace set of a DPN). The overall behavior of a DPN N with initial marking M_I and final marking M_F is the *set of complete finite process traces* that lead from the initial marking M_I to the final marking M_F :

$$TS_{N, M_I, M_F} = \{\sigma_N \in PS^* \mid \exists_{\alpha_F \in U_{dom}^P} ((M_I, \alpha_I) \xrightarrow{\sigma_N} (M_F, \alpha_F))\}.$$

We denote TS_{N, M_I, M_F} as the *trace set* of a DPN.

◊

Example 3.5 (Trace set of a DPN). Assume again that the DPN specified in Figure 3.3 together with its initial marking $M_I = ([src], \alpha_I)$ and its set of final markings ($M_F = ([snk], \alpha_F)$). The trace set of this DPN consists of all complete finite process traces that start with the initial marking, i.e., only a token in the place *src*, and end with the final marking, i.e., only a single token in the place *snk*. Thus, only complete finite process executions are part of the trace set TS_{N, M_I, M_F} . For example, the firing sequence $\langle(t_{tri}, (\text{data}_{\text{color}} \rightarrow \text{White})), (t_{Reg}, \emptyset), (\tau_2, \emptyset)\rangle \in TS_{N, M_I, M_F}$ is part of the trace set whereas the firing sequence $\langle(t_{tri}, (\text{data}_{\text{color}} \rightarrow \text{Yellow})), (t_{reg}, \emptyset), (\tau_2, \emptyset)\rangle \notin TS_{N, M_I, M_F}$ is not part of the trace set since the guard of the transition τ_2 is not fulfilled.

Some transitions in a DPN do not correspond to actual pieces of work and are only added for routing purposes. For example, transitions τ_1, \dots, τ_4 in Figure 3.3 are such routing transitions. Formally, there is no reason to distinguish such transitions from others. In practical terms, such routing transitions are characterized by not leaving any explicit trails in event logs. That is why these transitions are commonly referred to as *invisible transitions*. Analogous to our definition of a labeled trace set, we introduce a labeled DPN.

Definition 3.12 (Labeled DPN). Let $N = (P, T, F, V_P, dom, in, wr, gd)$ be a DPN. A *labeled DPN* is a triple $LN = (N, \lambda, \nu)$, in which:

- $\lambda : T \rightarrow (\Sigma \cup \{\tau\})$ is an *activity label function* that returns the observable activity name of a transition or τ for invisible *routing transitions*,
- $\nu : V_P \rightarrow V_L$ is a *variable label function* that returns the observable attribute name of a variable.

◇

Example 3.6 (Relating firing sequences to activities and attributes with a labeled DPN). Figure 3.4 shows a labeled version of the DPN N that was introduced in Figure 3.3. We depict both mapping functions λ and ν by annotating the DPN with the names of the observable activities and the names of the observable data attributes of the process. In this labeled DPN, we can connect transition of the DPN to activities recorded in an event log. In Figure 3.4 each transition is mapped to a unique activity. Generally, two or more transitions may be connected to the same activity. In a similar manner, we can connect the other attributes of the event log to variables in the DPN. For example, we specify that both variables res_{reg} and res_{pre} are mapped to the *resource* attribute in the event log and variable $time_{che}$ is mapped to the *time* attribute.

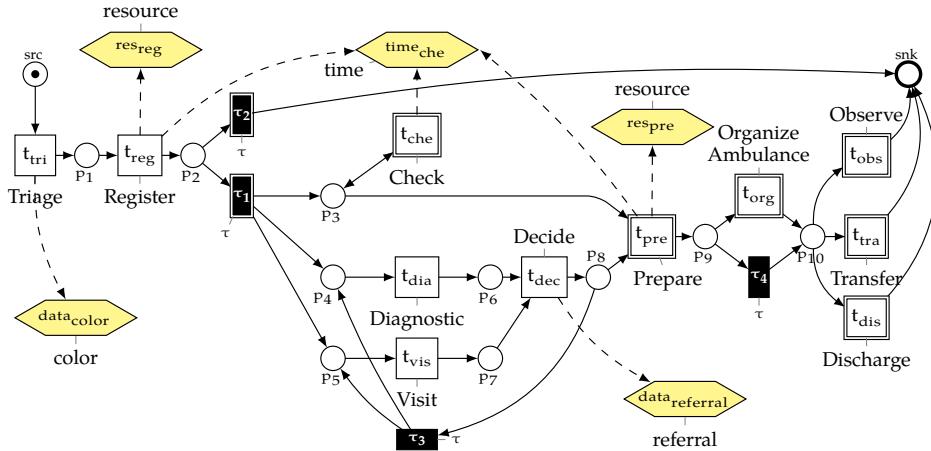


Figure 3.4: A labeled DPN describing the hospital process. Activity and variable labels are shown next to the model elements.

3.3 Causal Nets

We use *Causal Nets* (C-Nets) [AAD11; Aal16; WR11] as notation for the process discovery method presented in Chapter 8. Other notations similar to C-Nets have been proposed [AAD11], e.g., Heuristic Nets [WAA06; WR11] and Fuzzy Models [GA07]. Moreover, most commercial process mining tools use simple representations of the dependency relation that form the core of a C-Net. We use the syntax and semantics of C-Nets proposed by Van der Aalst et al. [AAD11; Aal16]. The C-Nets represent only the control-flow perspective of a process. Therefore, we present a multi-perspective extension in Chapter 8.

C-Nets are directed graphs that represent the *causal dependencies (dependency relations)* between activities. Nodes of the graph are *activities* of the process. Directed edges between two activities define a dependency relation, i.e., the execution of the target activity depends on the prior execution of the source activity. There are no routing elements in a C-Net. Instead, each activity is associated with a set of input bindings and output bindings. Input bindings define sets of activities that can precede an activity. Output bindings define sets of activities that can follow an activity.

Definition 3.13 (Causal Net [Aal16]). A C-Net is a 6-tuple $C = (\Sigma, a_i, a_o, D, I, O)$ where:

- Σ is a finite set of activities,
- $a_i \in \Sigma$ is the start activity,
- $a_o \in \Sigma$ is the end activity,

- $D \subseteq \Sigma \times \Sigma$ is the dependency relation,
- $AS = \{X \subseteq \mathbb{P}(\Sigma) \mid X = \{\emptyset\} \vee \emptyset \notin X\}$ are sets of activities,
- $I : \Sigma \rightarrow AS$ is the set of input bindings per activity, and
- $O : \Sigma \rightarrow AS$ is the set of output bindings per activity

such that the dependency relations match the input and output bindings:

$$D = \{(a_1, a_2) \in \Sigma \times \Sigma \mid a_1 \in \bigcup_{\beta \in I(a_2)} \beta \wedge a_2 \in \bigcup_{\beta \in O(a_1)} \beta\}$$

and that the C-net has a *unique* start and end activity, i.e., $\{a_i\} = \{a \in \Sigma \mid I(a) = \{\emptyset\}\}$ and $\{a_o\} = \{a \in \Sigma \mid O(a) = \{\emptyset\}\}$. \diamond

Each activity of a C-net refers to a unique activity of the process; thus, we can directly associate a recorded event in the event log to an activity in the C-net. However, C-nets might not include all activities that are recorded in the event log.

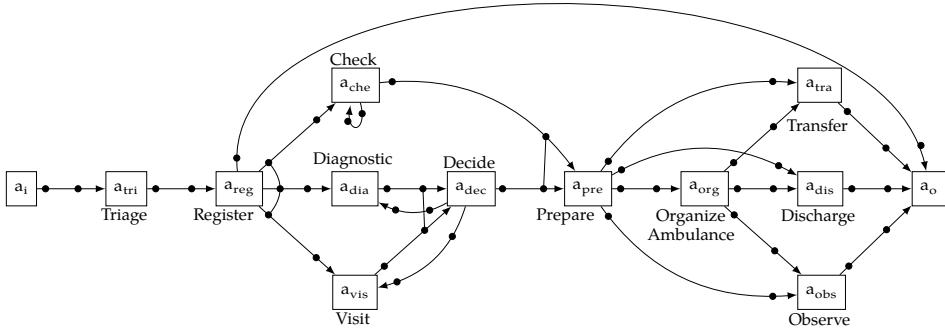


Figure 3.5: The hospital process modeled as C-Net. The bindings of the C-Net are depicted as black dots on the edges between activities.

Example 3.7 (Running example process modeled as a C-net). Figure 3.5 shows a C-net modeling the process behavior that is described for the hospital process in Example 1.1. Activities of the C-net are depicted as boxes. We refer to the activities by using a shorthand version, e.g., a_{tri} corresponds to the activity *Register*. Dependency relations of the C-net are shown as directed edges between activities. For example, the execution of activity a_{reg} (*Register*) depends on a prior execution of the activity a_{tri} (*Triage*). The C-net includes a unique start activity a_i and a unique end activity a_o .

The sets of input and output bindings of each activity obtained by the binding functions I and O are depicted using small black dots on the outgoing and incoming edges. Connected dots indicate that multiple activities are part of the same binding. For example, the set of output bindings of activity a_{tri} is $O(a_{tri}) = \{\{a_{reg}\}\}$

and its set of input bindings is $I(a_{tri}) = \{\{a_i\}\}$. Activities may have multiple input or output bindings, e.g., the set of output bindings of activity a_{che} (*Check*) is $O(a_{che}) = \{\{a_{che}\}, \{a_{pre}\}\}$. In a C-net, multiple possible bindings indicate an exclusive choice between the activities that are specified by the bindings. For example, after an occurrence of a_{che} , either a_{pre} may be executed or a_{che} may be executed again. Conversely, multiple possible bindings in an input binding indicate a simple merge.

A single binding may consist of multiple activities, i.e., function I and O return a set of sets for each activity. We depict bindings that consist of multiple activities by connecting the dots on the edges. For example, the set of output bindings of activity a_{tri} is $O(a_{tri}) = \{\{a_o\}, \{a_{vis}, a_{dia}, a_{che}\}\}$. The binding $\{a_{vis}, a_{dia}, a_{che}\}$ specifies a parallel split, i.e., all activities of the binding, i.e., a_{vis} , a_{dia} , and a_{che} , must occur (AND split). Conversely, input bindings that consists of multiple activities indicate a synchronization (AND join). In the output binding of a_{reg} both types are mixed, i.e., there is an exclusive choice between the single activity $\{a_o\}$ and the parallel split $\{a_{vis}, a_{dia}, a_{che}\}$. This corresponds to an inclusive choice, which is also called multi-choice or OR split. Figure 3.6 gives an overview of the kinds of routing constructs that can be modeled using bindings of a C-net.

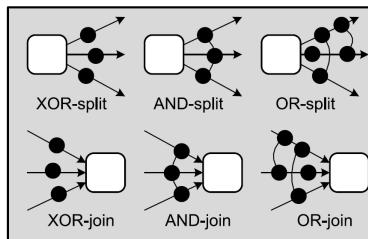


Figure 3.6: Possible routing constructs using a set of bindings in a C-net [Aal16].

We adopt the formal semantics introduced by Van der Aalst et al. [AAD11]. The execution semantics of C-Nets is defined through the interplay between the input and output bindings of activities. Only full execution traces are considered, i.e., the semantics are global in contrast to the local semantics of a DPN.

Definition 3.14 (Binding sequence [Aal16]). Let $C = (\Sigma, a_i, a_o, D, I, O)$ be a C-Net. We denote the set of possible activity bindings of C as:

$$B = \{(a, as^I, as^O) \in \Sigma \times \mathbb{P}(\Sigma) \times \mathbb{P}(\Sigma) \mid as^I \in I(a) \wedge as^O \in O(a)\}.$$

A binding sequence is a sequence of activity bindings: $\sigma \in B^*$. ◊

A binding sequence defines the set of activities (as^I) that are expected to occur before an activity a can be executed and the set of activities (as^O) that are expected

to occur after execution of \mathbf{a} . Intuitively, a pair of output and input bindings in a binding sequence matches if there are sets of activities in the output bindings and input bindings of activities such that the activity sets agree. Every activity that was *activated* by an output binding has to be *deactivated* by an activity in an input binding. We denote the activation of sets of activities as *obligations*. Output bindings add obligations and input bindings remove obligations.

A binding sequence $\langle (a_1, as_1^I, as_1^O), \dots, (a_n, as_n^I, as_n^O) \rangle$ is considered a *valid binding sequence* if the sequences starts with the start activity a_i ($a_0 = a_i$), ends with the end activity a_o ($a_n = a_o$), a_i and a_o do not occur in other places of the sequence, and the output bindings an input bindings of the activities match, i. e., there are no pending obligations left after the sequence ends.

Example 3.8 (Valid binding sequence). We show two examples for valid binding sequence. We underline the executed activity for improved legibility. The first example of a valid binding sequence is:

$$\sigma_1 = \langle (\underline{a_i}, \emptyset, \{a_{tri}\}), (\underline{a_{tri}}, \{a_i\}, \{a_{reg}\}), \\ (\underline{a_{reg}}, \{a_{tri}\}, \{a_o\}), (\underline{a_o}, \{a_{reg}\}, \emptyset) \rangle.$$

Binding sequence σ_1 is valid and corresponds to a process execution in which only the activities *Triage* (a_{tri}) and *Register* (a_{reg}) occur, i. e., the patient left before being admitted to the emergency ward. Start- and end activities a_i and a_o are artificially introduced activities that serve solely as markers for the boundary of the process, i. e., the input and output of the process. A second example of a valid binding sequence is:

$$\sigma_2 = \langle (\underline{a_i}, \emptyset, \{a_{tri}\}), \\ (\underline{a_{tri}}, \{a_i\}, \{a_{reg}\}), (\underline{a_{reg}}, \{a_{tri}\}, \{a_{vis}, a_{dia}, a_{che}\}), \\ (\underline{a_{che}}, \{a_{che}\}, \{a_{che}\}), (\underline{a_{che}}, \{a_{che}\}, \{a_{che}\}), (\underline{a_{che}}, \{a_{che}\}, \{a_{pre}\}), \\ (\underline{a_{vis}}, \{a_{reg}\}, \{a_{dec}\}), (\underline{a_{dia}}, \{a_{reg}\}, \{a_{dec}\}), \\ (\underline{a_{dec}}, \{a_{dia}, a_{vis}\}, \{a_{pre}\}), \\ (\underline{a_{pre}}, \{a_{dec}, a_{che}\}, \{a_{tra}\}), \\ (\underline{a_{tra}}, \{a_{pre}\}, \{a_o\}), \\ (\underline{a_o}, \{a_{tra}\}, \emptyset) \rangle.$$

This binding sequence corresponds to a process execution in which the patient is admitted to the emergency ward. The patient is checked three times, visited by the doctor, and a medical diagnostic test is taken. Then, the patient is prepared and transferred.

We formalize this execution semantics by introducing the *state* of a C-Net and, then, defining its behavior through the set of *valid binding sequences*.

Definition 3.15 (State of a C-Net [Aal16]). Let $C = (\Sigma, a_i, a_o, D, I, O)$ be a C-Net. The *state* of a C-Net $ST_C \in \mathbb{B}(A \times A)$ is a multiset of *pending obligations*. We define function $state_C : B^* \rightarrow ST_C$ that returns the state after executing binding sequence $state_C(\sigma)$ as:

- $state_C(\langle \rangle) = []$
- $state_C(\sigma \cdot (a, as^I, as^O)) = (state_C(\sigma) \setminus (as^I \times \{a\})) \uplus (\{a\} \times as^O)$

for any binding sequence $\sigma \cdot (a, as^I, as^O) \in B^*$. \diamond

The state of a C-Net is similar to a marking of a Petri net. Pending obligations can be seen as tokens, which need to be consumed by subsequent activities.

Definition 3.16 (Valid binding sequences of a C-Net [Aal16]). Let $C = (\Sigma, a_i, a_o, D, I, O)$ be a C-Net. Let $\sigma = \langle (a_1, as_1^I, as_1^O), \dots, (a_n, as_n^I, as_n^O) \rangle \in B^*$ be a binding sequence. Sequence σ is in the set of valid binding sequences BS_C of C if and only if:

1. $a_0 = a_i \wedge \forall_{1 < j < n} (a_j \in (\Sigma \setminus \{a_i, a_f\})) \wedge a_n = a_f$;
2. there are no pending obligations, i.e., $state_C(\sigma) = []$; and
3. for any prefix $\langle (a_1, as_1^I, as_1^O), \dots, (a_j, as_j^I, as_j^O) \rangle = \sigma' \cdot (a_j, as_j^I, as_j^O) \in pref(\sigma)$ we require that $(as_j^I \times \{as_j^O\}) \leq state_C(\sigma)$, i.e., bindings may only remove pending obligations for the current activity as_j^O . \diamond

Example 3.9 (Declarative replay semantics of a C-Net). Assume the C-Net shown in Figure 3.5. Binding sequence σ_1 (Example 3.8) is part of its valid binding sequences, i.e., $\sigma_1 \in BS_C$. Sequence σ_1 starts with a_i and ends with a_o . There are no pending obligations after replaying binding sequence σ_1 on the C-net. Moreover, for any prefix of σ_1 , only pending obligations are removed. The *replay* of the binding sequence changes the state of the C-Net as follows:

$$\begin{aligned} ST_C(\langle \rangle) &= [] \\ ST_C(\langle (a_i, \emptyset, \{a_{tri}\}) \rangle) &= [(a_i, a_{tri})] \\ ST_C(\langle (a_i, \emptyset, \{a_{tri}\}), (a_{tri}, \{a_i\}, \{a_{reg}\}) \rangle) &= [(a_{tri}, a_{reg})] \\ ST_C(\langle (a_i, \emptyset, \{a_{tri}\}), (a_{tri}, \{a_i\}, \{a_{reg}\}), (a_{reg}, \{a_{tri}\}, \{a_o\}) \rangle) &= [(a_{reg}, a_o)] \\ ST_C(\langle (a_i, \emptyset, \{a_{tri}\}), (a_{tri}, \{a_i\}, \{a_{reg}\}), (a_{reg}, \{a_{tri}\}, \{a_o\}), (a_o, \{a_{reg}\}, \emptyset) \rangle) &= []. \end{aligned}$$

We show that C-Nets can be used to specify the set of execution traces, i.e., analogous to the trace set of a DPN (Definition 3.11), we define the overall trace set of a C-Net.

Definition 3.17 (Trace set of a C-Net). Let $C = (\Sigma, a_i, a_o, D, I, O)$ be a C-Net. Let $V_P \subseteq V$ be a set of process variables. Let T be a set of process transitions such that

$T = \Sigma$, i. e., we assume that the activities of the C-Net are process transitions. Let $PS = T^*$ be the set of all possible process steps. The overall behavior of the C-Net C is the set of process traces with a corresponding valid binding sequence in C :

$$\begin{aligned} TS_C = \{ & \langle (t_1, \emptyset), \dots, (t_n, \emptyset) \rangle \in PS^* \mid \exists_{\sigma \in BS_C} (\sigma = \langle (a_i, as_0^I, as_0^O), \\ & (t_1, as_1^I, as_1^O), \dots, (t_n, as_n^I, as_n^O), \\ & (a_o, as_{n+1}^I, as_{n+1}^O) \rangle) \}. \end{aligned}$$

We denote TS_C as the trace set of C . \diamond

Since C-Nets are defined directly over the same set of activities Σ that is recorded in event logs, we do not need to introduce a labeled variants.

3.4 BPMN and Extended Data Petri Nets

In this section, we show that the process modeling notations that we presented in the previous three sections are compatible with notations that provide a higher level of abstraction. Since we only want to illustrate the compatibility with the notations used in this thesis, both BPMN and eDPN are introduced by means of examples.

3.4.1 BPMN

BPMN [BPMN11] is a process modeling notation that has become the de-facto standard in practice. It is widely supported by tool vendors (e. g., Activiti, Bonita BPM, Camunda, IBM Rational System Architect, and Signavio), used in text books [Dum+13], and has been adopted by many organizations [Rec10]. BPMN is a procedural process notation with token-based semantics. A commonly subset of BPMN [MR08] has been formalized, e. g., by providing a mapping to Petri nets [DDO08]. Conversely, there are multiple process discovery methods that first discover one of the previously introduce notations (i. e., Petri nets, DPNs, and Causal Nets) and, subsequently, transform those discovered models to BPMN models [Con+16; Kal+16a; Kal+16b; WBC14]. Thus, the mapping between the notations used in this thesis and BPMN is well-researched and supported by tools like ProM. As an example of a transformation, Figure 3.7 shows the multi-perspective BPMN model of the hospital process transformed from the DPN in Figure 3.4.

The BPMN process starts with a start event (circle) and ends with an end event (thick circle). The transitions of the DPN are transformed into BPMN activities, which are depicted as rounded rectangles. BPMN supports special activities with a loop marker (\circlearrowright) that can be repeated indefinitely [BPMN11]. In Figure 3.7, we use such a loop marker to express that activity *Check* can be repeated in the same manner as expressed by the Petri net in Figure 3.2 and the C-Net in Figure 3.5. Since BPMN

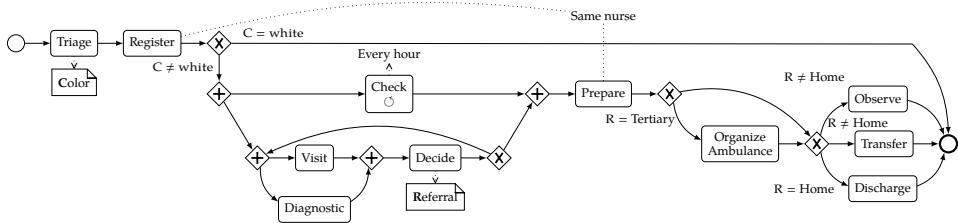


Figure 3.7: A BPMN model of the hospital process.

does not support the notation of a state in terms of marked places like in a DPN, gateways need to be used as additional modeling elements. In Figure 3.4, there are two types of gateways: exclusive choice gateways (\times) and parallel gateways (+). We transform guards of a DPN into data conditions that are placed on exclusive gateways. For example, the guard $\text{data}_{\text{color}} = \text{white}$ for patients, who leave the hospital without being admitted, is transformed into a data condition written above the edge from the first exclusive gateway to the end event. Variables of the DPN are transformed to data objects, which are connected to activities that assign values. In Figure 3.4, we use simple textual annotations to specify the resource and time rules of the hospital process. BPMN offers more sophisticated support for modeling temporal aspects and some limited support for modeling resources (e.g., through swimlanes as depicted in Figure 1.6), but a comprehensive introduction would be out of the scope in this thesis.

3.4.2 Extended Data Petri Nets

Extended Data Petri Nets (eDPNs), introduced by Balkom [Bal16], extend the DPN notation with additional notational elements that describe high-level multi-perspective constructs. The eDPN notation aims at simplifying the notation of those complex multi-perspective constructs at the expense of introducing additional elements to the language, which express time and resource-perspective related constraints. Multi-perspective process models can be visualized in a more compact form. The elements of eDPN are inspired by research on the visualization of multi-perspective compliance rules in the context of the extended Compliance Rule Graph (eCRG) language [KR16], which has been evaluated with users and shown to be applicable in real-world situations [KR16; KRK17].

The notational elements introduced by the eDPN notation are defined by a mapping to DPN constructs (i.e., similar to macros in programming languages). Figure 3.8 shows the hospital process modeled in eDPN notation. The DPN in Figure 3.3 uses multiple guard expressions to encode the time-perspective constraint that activity *Check* should be executed every hour from registration of the patient until preparation for discharge or transfer. In the eDPN notation these guards are

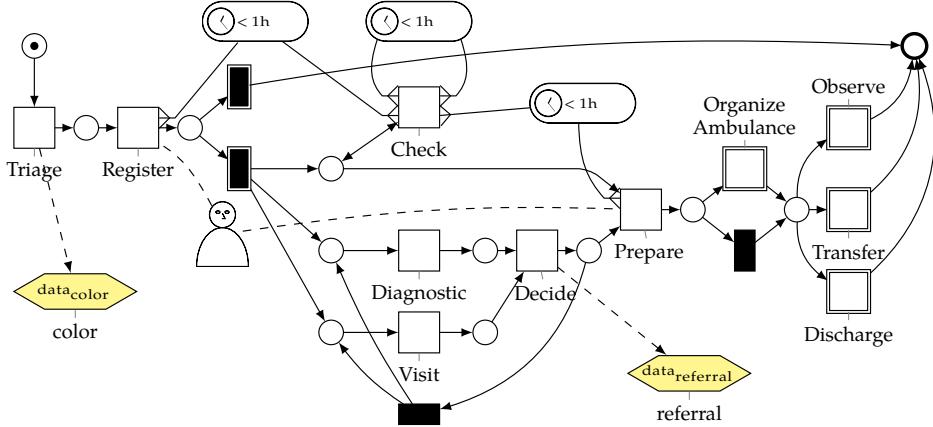


Figure 3.8: An eDPN model of the hospital process.

replaced by a visual *time distance constraint* element that is connected to both activities for which a time constraint is defined. Similarly, the guards encoding the *retain familiar resource* constraint are also replaced by a new visual element. After conversion of the eDPN to a DPN as defined in [Bal16] the methods proposed in this thesis can be applied.

Part II

Multi-perspective Conformance

Chapter 4 We position multi-perspective conformance checking in the broader context of compliance checking and process mining. Moreover, we introduce the central concept of an alignment. An alignment establishes a mapping between a log trace of the event log and a process trace of the trace set of a process model.

Chapter 5 We present a method that computes an alignment with the goal to balance deviations in multiple perspectives on the process to provide reliable conformance diagnostics. Moreover, we show that the balanced multi-perspective alignment method can be used to quantify the level of fitness between the observed and modeled behavior. This chapter is based on the publications [Man+14; Man+16a].

Chapter 6 We present a method to quantify the precision of modeled behavior based on observed behavior in form of an event log. The precision measure makes use of the balanced multi-perspective alignment introduced in Chapter 5. This chapter is based on the publication [Man+16d].

4 Introduction to Multi-perspective Conformance Checking

Organizations maintain *process models* to describe their business processes. Process models may be manually modeled or obtained through process discovery. Irregardless of its source a process model should accurately reflect the real execution of the process, i. e., it should reflect the *process reality*.

Rules and regulations (e. g., Sarbanes-Oxley Act [SOX02], Basel II [BASEL06], and the ISO 9000 series [ISO15]) may require organizations to document their operational or reporting processes accurately [KMS07]. When the documentation in the form of a process model and the process reality *deviate*, then, either the documentation is outdated and the process performs well, or, worse, the real execution of the process does not follow the desired procedures with possible harmful consequences. Thus, inaccurate process models are unsuited for their prime purposes: documentation and analysis of processes. Conversely, processes that do not follow the desired procedures may lead to bad performance or form a risk to the whole organization in case of non-compliance with rules and regulations.

Considerable work has been devoted to checking the compliance of processes and process models with rules and regulations. The work can be categorized in two categories: *forward compliance checking* and *backward compliance checking* [Kha+08]. Forward compliance checking methods focus on verifying the compliance of process models to rules at the design time (e. g., [ADW08]) and on monitoring the adherence to rules during the execution of a processes (e. g., [Ly+15]). *Backward compliance checking* employs data recorded about the process execution to locate compliance violations, e. g., in the context of audits [Aal+10; AS12; Car13].

This chapter presents methods that can be categorized as *backwards compliance checking* methods, i. e., we analyze the **historic process execution**. However, we do not focus solely on checking the compliance of a process execution to rules and regulations. We consider the problem of *comparing* modeled and observed behavior, *quantifying* the correspondence, providing *diagnosis information* for process executions based on process models, and analyzing bottlenecks and resource usage. In the next section, we introduce the field of conformance checking, which encompasses all those problems.

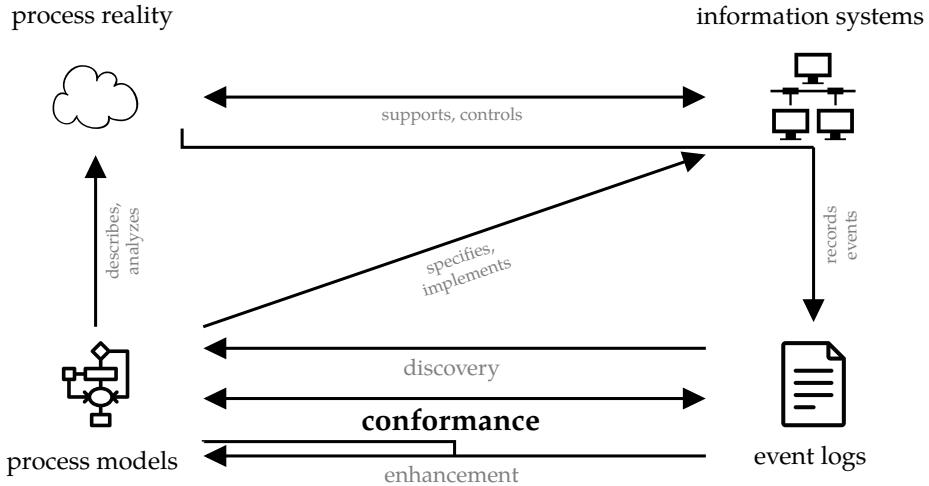


Figure 4.1: Conformance checking of processes in the context of process mining [Aal16].

4.1 Conformance Checking

Conformance checking is the diagnosis and quantification of discrepancies between process reality, i. e., the real execution recorded by information systems and process models, i. e., the desired execution of the process. Information systems often record information about the execution of process activities in their databases. This information can be extracted from a database in the form of an event log [Aal16]. As shown in Figure 4.1, recorded event logs are then used by process conformance checking methods together with existing or discovered process models as input to compare the real process execution with the assumed behavior specified by process models.

Many methods have been proposed that address the problem of *comparing observed and modeled behavior* in the context of business processes. We differentiate between *local conformance checking methods* and *global conformance checking methods*.

Local conformance checking methods consider a set of independent rules as input for conformance checking. Each rule expresses constraints for a part of the overall process. For example two simple rules in the context of our hospital example process could be:

- executions of *Diagnostic* should precede executions of *Decide* (A) and
- executions of *Check* should precede executions of *Prepare* (B).

Rule A is fulfilled in the log trace that is shown in Figure 4.2: event *Diagnostic* occurs before event *Decide*. Rule B is violated since event *Check* does not occur before event *Prepare*. The adherence to such rules can be verified very efficiently for

each rule in isolation. Rules are often specified in some kind of temporal logic (e.g., Linear Temporal Logic (LTL) [ABD05; Car13]) or in a declarative process model notation. Most declarative notations are grounded in a suitable temporal logic (e.g., Declare [PSA07] or DCR Graphs [HMS12]) or assigned a precise semantics with other means (e.g., Compliance Rule Graphs [Ly+11] or Petri net patterns [RFA12]). Clearly, more intricate rules than the two examples given can be verified. The diagnosis is often limited to a *binary* output: the rule is *fulfilled* or the rule is *not fulfilled*.

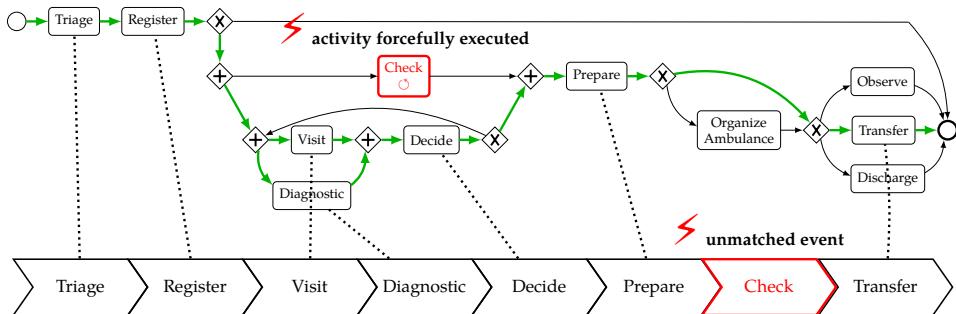


Figure 4.2: Global conformance checking methods establish a global connection between the events and elements of the process model.

Global conformance checking methods expect an integrated end-to-end, often procedural, process model as input, e.g., a simple BPMN model that is shown in the top part of Figure 4.2. The process model is expected to be an accurate specification of the overall behavior of the process, i.e., a closed-world is assumed. Global conformance checking methods determine deviations between the observed ordering of activities and the expected ordering of activities. For example, in Figure 4.2 activity *Check* is diagnosed to have been forcefully executed. The *check event* was *missing* when it was expected, i.e., after activity *Register* and before activity *Prepare*. Moreover, later in the trace an extra *check event* occurred, but cannot be matched to a corresponding activity in that stage of the process.

Often, it is desirable to not only report the existence of conformance problems but also diagnostic information that enables to pinpoint the exact source of a deviations and analyze contextual factors. Moreover, there may be two interpretations of a deviation:

- The process model might not reflect the real execution properly, e.g., because it is outdated, based on optimistic scenarios, or not flexible enough to capture all scenarios. Therefore, the process model should be improved.
- The process execution might not adhere to rules imposed by a normative process model. Non-conformance may lead to poor performance or, in case of non-compliance to regulations, to legal problems. However, non-conformance

may also be a positive deviation from the prescribed process, i. e., a different handling of cases may result in better quality outcomes or faster processing.

Besides compliance-oriented questions, there are two more important aspects of conformance checking to the field of process mining [Aal+12]. Conformance checking methods:

- establish the relation between events in the event log to elements in the process model (e. g., to transitions in a DPN based on an alignment as will be introduced in Section 4.3) and
- determine the quality of discovered process models.

As motivated in the previous paragraph, model-based methods establish a relation between events and elements in the model (cf., Figure 4.2). Moreover, model-based methods are particularly suited to diagnose the quality of discovered process models, which are expected to describe the overall behavior of an observed process. The quality of a process model can be looked upon from various perspectives. Broadly speaking, quality measures can be distinguished into *structural quality measures* and *behavioral quality measures*. Structural measures are concerned with properties such as model size, density, cyclicity, and structuredness [Men08; Pol12]. Structural measures are typically based solely on the structure of the process model itself; thus, the behavior and its connection to the underlying process reality is not taken into account. Conversely, behavioral measures only take the behavior (i. e., state-space) of the model into account. In the context of process mining four major quality dimensions for discovered process models have been proposed [Aal16; BDA14; Roz+08]:

- fitness, the degree to which the model **allows all** the observed behavior;
- precision, the degree to which the model **only allows** the observed behavior;
- generalization, the degree to which the model **generalizes** from the observed behavior; and
- simplicity, the degree to which the model is **not be needlessly complex**⁶.

Simplicity is often measured based on simple structural properties (e. g., model size [BDA14]). The other three quality criteria in process mining are clearly behavioral measures. Fitness and precision are only related to the process model and the event log at hand, whereas the generalization measure introduces a third element: the (unknown) real process system. Figure 4.3 illustrates the behavioral aspects of the three quality dimensions fitness, precision, and generalization in a Venn diagram. The diagram in Figure 4.3 compares the behavior specified by the process model with both the behavior observed in the event log and the actual behavior of the real process (i. e. the system under examination). We adopt the view of Buijs et al. [BDA14] who defines fitness, precision, and generalization in terms of the

⁶The principle that simpler models are to be preferred when all other criteria are mostly equivalent is often denoted as Occam's razor.

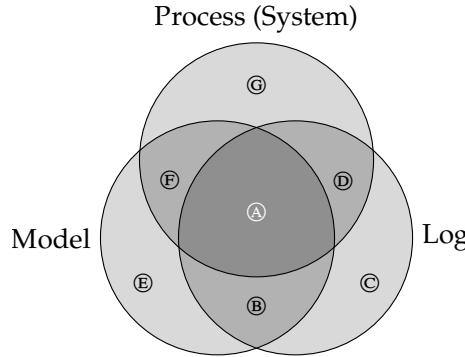


Figure 4.3: Venn diagram depicting the relation between the behavior specified by the process model, the behavior observed in the event log, and the possible behavior of the process (often denoted as system)[BDA14]. Note that the the behavior of the event log that is not part of the behavior of the process ($\textcircled{B} + \textcircled{C}$) is assumed to small since the event log contains recorded executions of the process.

overlaps between behaviors of model, log, and system.

Fitness can be expressed as the fraction of log behavior $\textcircled{A} + \textcircled{B}$ shared with the model over the overall observed log behavior $\textcircled{A} + \textcircled{B} + \textcircled{C} + \textcircled{D}$: $\frac{\textcircled{A} + \textcircled{B}}{\textcircled{A} + \textcircled{B} + \textcircled{C} + \textcircled{D}}$. *Precision* can be expressed as the fraction of model behavior $\textcircled{A} + \textcircled{B}$ shared with the log over the overall possible model behavior $\textcircled{A} + \textcircled{B} + \textcircled{E} + \textcircled{F}$: $\frac{\textcircled{A} + \textcircled{B}}{\textcircled{A} + \textcircled{B} + \textcircled{E} + \textcircled{F}}$. The problem with generalization is that the process system behavior is generally unknown in a process mining setting.⁷ Generalization measures try to estimate the size of area \textcircled{F} in Figure 4.3, i. e., how much unrecorded process system behavior is explained by the model [AAD12; Bro+14]. Area \textcircled{F} represents behavior that is not part of the event log, is described by the model, and is part of the system behavior. Thus, the model generalized over what was observed in the event log. We focus on fitness and precision measures since there is no generally agreed-upon measure for generalization [Jan+16].

In this section, we introduced the field of conformance checking and four quality measures for process models with regard to to event logs focusing on the control-flow perspective. In the next section, we extend conformance checking towards multiple process perspectives.

⁷In case the system behavior is known, e. g., the event log is known to be complete, then, there is little reason to use the generalization quality dimension.

4.2 Multi-perspective Conformance Checking

The major share of work on conformance checking techniques has focused only on the *control-flow perspective* (e. g., [AAD12; ADA11a; Adr14; Bro+14; CW99; MAW08; MC10; RA08; Roz10; Wei+11]). This means that, as illustrated in Figure 4.2, the order of steps is being analyzed to determine the conformance between prescribed and actual behavior. However, event logs often contain more information than just the order of activities. Additional elements, such as the processed data, information on organizational resources, and the execution timestamp are recorded. In fact, this additional information is as important for compliance as the ordering of activities [Awa10; Car13; CVB13b; Elg+14; Knu+10; RFA12; SGN07; Tur+12]. Recently, more work has been devoted to conformance checking for other important perspectives on processes: data, resources, and time; which may be also subject to rules and regulations (e. g., [BB14; BMS16; Car13; CVB13b; LA13a; Ram17]).

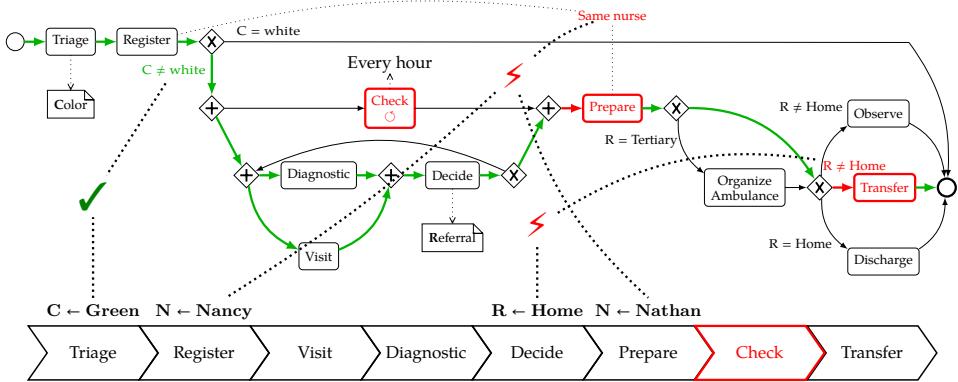


Figure 4.4: Model-based multi-perspective conformance checking needs to consider additional constraints based on data, resources, and time.

Multi-perspective conformance checking considers the conformance of process to multiple perspectives of a process model (i. e., control-flow, data, resources, time) at the same time. Deviations may occur on a perspective different from control-flow or deviations from the control-flow may be explained in terms of the other perspectives.

Multiple perspectives on a process are integrated in a single model and the sources for deviations are not only the ordering of activities, but also data-related deviations (e. g., the credit limit is too high), resource-related deviations (e. g., the four-eyes principle is violated), and time-related deviations (e. g., an activity is executed too late). If conformance checking would only consider the *control-flow perspective*, the activities themselves and their ordering are the only issues of concern. Executions may seem conforming to the model when, in fact, they are not.

However, to fully grasp whether a model conforms with reality other perspectives is important, as well. In this thesis, we focus on the additional perspectives data, resource, and time. For example, in Figure 4.4 rules based on contextual information and the control-flow prescribed by the model are checked using the recorded events.

Our multi-perspective process mining techniques are grounded in the belief that even though data, resource, and time are separate concerns from a business or modeling perspective, they can be encoded into the data perspective. For example, we can encode the *retain familiar* constraint [Rus+05] defined on the resource perspective of the model in Figure 4.4, i. e., both Register and Prepare should be done by the same doctor, using the data attribute **N**. Also the time constraints that activity *Check* needs to be executed every hour can be encoded using data attribute and rules, cf. the DPN specification of the process in Figure 3.3. We can, then, check conformance to the rules defined over attributes encoding the other perspectives. For example, in Figure 4.4, the *retain familiar* constraint is violated, activities *Register* and *Prepare* are executed by different resources *Nancy* and *Nathan*. Thus, we use the data perspective to capture any perspective different from control-flow. Whereas we only show such an encoding for resource and time dimensions, also other perspectives such as costs and risks can be encoded in the same way.

Conformance checking techniques that only consider the control-flow perspective *cannot* find the conformance violation of the *retain familiar* constraints for the trace depicted in Figure 4.4. By considering more perspectives more deviations between the process model and this trace can be identified. The identification of non-conformance in its different forms clearly has value in itself. For example, we can define quality measures *fitness* and *precision* that take multiple perspectives on the process into account. Nonetheless, organizations are often interested in explanations that can steer measures to improve the quality of the process. What may happen is that alternative explanations exist for a deviating trace. Due to the interplay among perspectives, different explanations may favor deviations in one perspective over deviations in another perspective.

For the identified deviation regarding the activity *Transfer* in Figure 4.4 there are two possible explanations:

1. The data value for attribute **R** was written correctly (i. e., the data-perspective is correct) but that activity *Transfer* should not have been performed (i. e., the control-flow perspective is incorrect). The patient should have been discharged ($R = \text{Home}$), but was wrongly transferred.
2. Activity *Transfer* was performed properly but attribute **R** was written incorrectly. The patient should have, indeed, been transferred and a different data value, e. g., $R = \text{Ward}$, should have been recorded.

Thus, apart from identifying multi-perspective deviations, it is desirable to identify possible explanations for the deviations. Having identified the most likely expla-

nation, it can be used, e. g., to investigate the root cause and project performance statistics on the model.

In this section, we motivated why it is crucial to consider multiple process perspectives for conformance checking of process models with regard to event logs. We introduced the problem of deriving the most likely explanation of deviations in terms of all process perspectives. Such an *explanation* of deviations between the prescribed behavior by the model and the observed behavior in the event log is denoted as an *alignment*. In the next section, we define multi-perspective alignments and introduce formal notations that are necessary for the presentation of our conformance checking methods.

4.3 Aligning Process Models and Event Logs

An *alignment* is a mapping between the modeled behavior in terms of a process model and the observed behavior from a real process execution recorded in an event log. Alignments are the central concept for model-based conformance checking that we use to check whether the process reality conforms to the models. The term alignment in process mining was coined by Adriansyah et al. [AAD12; Adr14] for the control-flow perspective. In this section, we introduce a broader multi-perspective definition of alignments inspired by the one presented by de Leoni et al. in [LA13a].

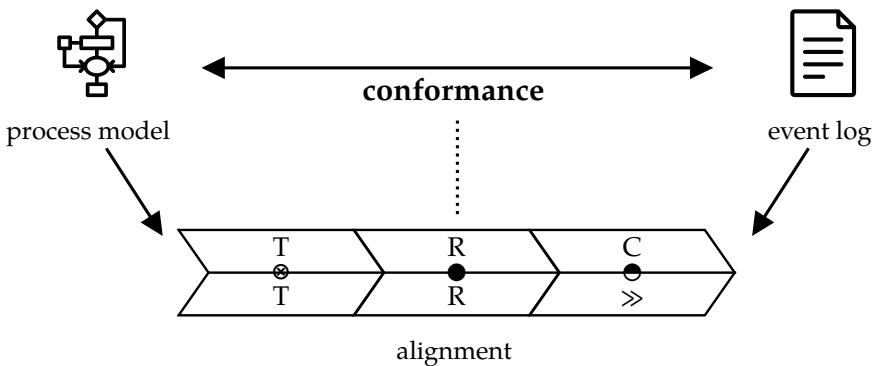


Figure 4.5: An alignment maps events to activities of a process model.

4.3.1 Alignments

We define alignments **independently of the modeling notation** based on an event log and a trace set. For this purpose, we use the trace set notation introduced in

Section 3.1, which is only based on the sets of labeled traces of a process model. An *alignment* establishes a mapping between a log trace of the event log and a process trace of the trace set. Events in the log trace (upper part of the events in Figure 4.5) are mapped to process steps in the process traces (lower part of the events in Figure 4.5). However, it may not be possible to align all events and process transitions. A special mismatch symbol \gg is used to express that there is a mismatch between log trace and process trace.

In case an event is mapped to \gg , the alignment could not find a corresponding process transition for an event, i. e., *the event is missing in the process model*. In case a process transition is mapped to \gg , the alignment could not find a corresponding event in the event log, i. e., *the process step was forcefully executed*. We denote those steps as *log move* and *model move*, respectively. Moreover, when considering the multiple perspectives, it is possible that the activity label of both the event and the process transition match, but the expected values of the process variables and values of the event attributes differ. We denote such a move as an *incorrect synchronous move*. Otherwise, if the expected values match, the move is denoted as a *correct synchronous move*.

Definition 4.1 (Alignment moves). Let $L = (E, \Sigma, \#, \mathcal{E})$ be an event log. Let PS be the set of all process steps. Let $LTS = (TS, \lambda, \nu)$ be a labeled trace set with $TS \subseteq PS^*$. A *legal move* in an alignment is a pair $(e, s) \in (E \cup \{\gg\}) \times (PS \cup \{\gg\})$ with:

- (e, \gg) is a *log move* (\ominus) iff $e \in E$,
- (\gg, s) is a *model move* (\bullet) iff $s \in S$,
- (e, s) is an *incorrect synchronous move* (\otimes) iff $e \in E \wedge s \in PS \wedge s = (t, w) \wedge \#_{act}(e) = \lambda(t) \wedge \exists_{v \in dom(w)}(\nu(v) \notin dom(\#(e))) \vee w(v) \neq \#_{\nu(v)}(e)$,
- (e, s) is a *correct synchronous move* (\bullet) iff $e \in E \wedge s \in PS \wedge s = (t, w) \wedge \#_{act}(e) = \lambda(t) \wedge \forall_{v \in dom(w)}(w(v) = \#_{\nu(v)}(e))$.

All other moves are considered illegal; e. g., the move (\gg, \gg) is an *illegal move*. We denote the *set of all legal moves* in an alignment as $\Gamma_{L,LTS} = ((E \cup \{\gg\}) \times (PS \cup \{\gg\})) \setminus \{(\gg, \gg)\}$. \diamond

Remember that $proj(\sigma, Y)$ returns the projection of sequence σ on set Y , i. e., only elements contained in Y are retained in the sequence. Given a sequence of alignment moves $\gamma = \langle (e_1, s_1), \dots, (e_n, s_n) \rangle \in \Gamma_{L,LTS}^*$. We introduce two notations on sequences of alignment moves:

- $proj_L(\gamma) = proj(\langle e_1, \dots, e_n \rangle, E)$ denotes the **log projection** of γ , i. e., the sequence of events ignoring occurrences of \gg .
- $proj_P(\gamma) = proj(\langle s_1, \dots, s_n \rangle, PS)$ denotes the **process projection** of γ , i. e., the sequence of process steps ignoring occurrences of \gg .

Definition 4.2 (Alignment). Let $L = (E, \Sigma, \#, \mathcal{E})$ be an event log. Let $LTS = (TS, \lambda, \nu)$ be a labeled trace set. An *alignment* between log trace $\sigma \in \mathcal{E}$ and labeled trace set LTS is a sequence of alignment moves $\gamma_\sigma \in \Gamma_{L,LTS}^*$ such that ignoring all occurrences of \gg , the projection of γ_σ on the first element (*log projection*) of each move yields log trace σ and the projection on the second element (*process projection*) yields a process trace. We define the set of alignments between σ and LTS as:

$$\Gamma_{L,LTS}^\sigma = \{\gamma_\sigma \in \Gamma_{L,LTS}^* \mid proj_L(\gamma_\sigma) = \sigma \wedge proj_P(\gamma_\sigma) \in TS\}.$$

An alignment $\gamma_\sigma \in \Gamma_{L,LTS}^\sigma$ is a sequence of alignment moves, which relate events of the log trace σ to process steps of a process trace in the process model. \diamond

Example 4.1 (Alignment). Table 4.1 shows three alignments between traces of the example event log (cf., Example 2.4) and the hospital process (cf., Figure 3.4).

Correct Synchronous Moves (●). The first two alignment moves in Table 4.1a are correct synchronous moves, i. e., the recorded events and the behavior prescribed by the model agree. For example, the DPN-guard color \neq White attached to the routing transition τ_1 is fulfilled since the triage color is recorded as *Red*. The third move $(\gg, (\tau_1, \emptyset))$ is a model move since invisible routing transition are never recorded in event logs.

Incorrect Synchronous Move (⊗). The seventh move $(e_{25}, (t_{che}, w_{25}))$ in Table 4.1a is an incorrect synchronous move. The activity in the event log can be mapped to the process transition in the DPN. However, the time constraint for activity *Check* (“[...] the nurse periodically checks their condition every hour”) requires event e_{25} to occur within one hour from *Register*. The *time* attribute value for both activity *Register* and *Check* need to be aligned to the values of the process variable $time_{che}$. Event e_{25} recorded an execution time of 5/12/16 22:25. This violates the time rule since it is more than one hour after the execution time of the *Register* activity (event e_{22} at 5/12/16 21:15). Hence, one of the values needs to be *corrected*. Here, an alternative, feasible value for the seventh move is suggested and it is marked as incorrect synchronous move.

Log Move (◐). Both events e_{40} (*Triage*) and e_{42} (*Check*) in the alignment Table 4.1b are problematic. According to the process model (Figure 3.4), *Check* is only done for patients that are admitted. One possible alignment of the log trace is to mark event e_{42} as log move, i. e., the event is invalid and cannot be explained by the process model and the value recorded for attribute *color* as incorrect.

Table 4.1: Three alignments between log traces $\sigma_2, \sigma_4 \in \mathcal{E}_{\text{ex}}$ and the hospital process.(a) γ_{σ_2} : One incorrect synchronous move (\otimes), two model moves (\bullet), and one log move (\circlearrowleft).

Event attributes (#(e))	Move (e, (t, w))	Process variables (w)
(activity \mapsto Triage, color \mapsto Red)	\bullet (e ₂₀ , (t _{tri} , w ₂₀))	(data _{color} \mapsto Red)
(activity \mapsto Register, time \mapsto 5/12/16 21:02, resource \mapsto Nancy)	\bullet (e ₂₁ , (t _{reg} , w ₂₁))	(time _{che} \mapsto 5/12/16 21:02, res _{reg} \mapsto Nancy)
-	\bullet (\gg , (τ_1 , \emptyset))	\emptyset
(activity \mapsto Check, time \mapsto 5/12/16 21:15)	\bullet (e ₂₂ , (t _{che} , w ₂₂))	(time _{che} \mapsto 5/12/16 21:15)
(activity \mapsto Diagnostic)	\bullet (e ₂₃ , (t _{dia} , \emptyset))	\emptyset
(activity \mapsto Visit)	\bullet (e ₂₄ , (t _{vis} , \emptyset))	\emptyset
(activity \mapsto Check, time \mapsto 5/12/16 22:25)	\otimes (e ₂₅ , (t _{che} , w ₂₅))	(time _{che} \mapsto 5/12/16 22:15)
(activity \mapsto Decide, referral \mapsto Tertiary)	\bullet (e ₂₆ , (t _{dec} , w ₂₆))	(data _{referral} \mapsto Tertiary)
(activity \mapsto Prepare, time \mapsto 5/12/16 22:32, resource \mapsto Nancy)	\bullet (e ₂₇ , (t _{pre} , w ₂₇))	(time _{che} \mapsto 5/12/16 22:32, res _{pre} \mapsto Nancy)
(activity \mapsto Organize Ambulance)	\bullet (e ₂₈ , (t _{org} , \emptyset))	\emptyset
(activity \mapsto Discharge)	\bullet (e ₂₉ , \gg)	\emptyset
-	\bullet (\gg , (t _{tra} , \emptyset))	\emptyset

(b) γ_{σ_4} : 1 incorrect synchronous move (\otimes), one model move (\bullet), and one log move (\circlearrowleft).

Event attributes (#(e))	Move (e, (t, w))	Process variables (w)
(activity \mapsto Triage, color \mapsto Green)	\otimes (e ₄₀ , (t _{tri} , w ₄₀))	(data _{color} \mapsto White)
(activity \mapsto Register, time \mapsto 15/12/16 10:45, resource \mapsto Nathan)	\bullet (e ₄₁ , (t _{reg} , w ₄₁))	(time _{reg} \mapsto 15/12/16 10:45, res _{reg} \mapsto Nathan)
-	\bullet (\gg , (τ_2 , \emptyset))	\emptyset
(activity \mapsto Check)	\bullet (e ₄₂ , \gg)	\emptyset

(c) An alternative alignment $\bar{\gamma}_{\sigma_4}$ for trace σ_4 containing six model moves (\bullet).

Event attributes (#(e))	Move (e, (t, w))	Process variables (w)
(activity \mapsto Triage, color \mapsto Green)	\bullet (e ₄₀ , (t _{tri} , \bar{w}_{40}))	(data _{color} \mapsto Green)
(activity \mapsto Register, time \mapsto 15/12/16 10:45, resource \mapsto Nathan)	\bullet (e ₄₁ , (t _{reg} , \bar{w}_{41}))	(time _{reg} \mapsto 15/12/16 10:45, res _{reg} \mapsto Nathan)
(activity \mapsto Check, time \mapsto 15/12/16 11:15)	\bullet (e ₄₂ , (t _{che} , \bar{w}_{42}))	(time _{che} \mapsto 15/12/16 11:15)
-	\bullet (\gg , (t _{dia} , \emptyset))	\emptyset
-	\bullet (\gg , (t _{vis} , \emptyset))	\emptyset
-	\bullet (\gg , (t _{dec} , \bar{w}_{45}))	(data _{referral} \mapsto Home)
-	\bullet (\gg , (t _{pre} , \bar{w}_{46}))	(time _{che} \mapsto 15/12/16 11:15, res _{pre} \mapsto Nathan)
-	\bullet (\gg , (τ_4 , \emptyset))	\emptyset
-	\bullet (\gg , (t _{tra} , \emptyset))	\emptyset

Model Move (O). The alignment shown in Table 4.1c gives a different explanation of the deviations observed in trace σ_4 . Instead of assuming that the event e_{42} (*Check*) was wrongfully recorded, five process steps have been inserted in order to finish the process trace. Note that the values assigned to the process variables of these model moves need to be feasible solutions to the constraints imposed by DPN-guards.

4.3.2 Optimal Alignment and Cost Function

Some alignments may be more desirable or likely than others. For example, alignment $\bar{\gamma}_{\sigma_4}$ (Table 4.1c) indicates that the events of six process transitions have not been recorded in trace σ_4 . Alignment γ_{σ_4} diagnosed only three deviations: one incorrect synchronous move, one model move, and one log move. Both alignments are valid according to Definition 4.2 and it depends on the application scenario which of both is preferable. In the case of σ_4 there are two plausible scenarios:

1. We know that the recording of attributes values is manual and known to be error-prone. Moreover, we know that the process instance recording σ_4 has been completed.
2. We know that attribute values are automatically recorded and known to be correct.

In the first scenario, alignment $\bar{\gamma}_{\sigma_4}$ seems to be very unlikely since it suggest that four activities in a row have not been recorded. However, a incorrectly recorded triage color as diagnosed by alignment γ_{σ_4} could be a plausible explanation. In the second scenario, alignment $\bar{\gamma}_{\sigma_4}$ could be a good explanation. It diagnoses that the process instance recording σ_4 is still running and the alignment indicates a possible continuation.

Often, the choice which alignment is better can only be made using domain knowledge. Therefore, we use a domain-specific cost function to rank different alignments of the same trace.

Definition 4.3 (Cost function). Let $L = (E, \Sigma, \#, \mathcal{E})$ be an event log. Let $LTS = (TS, \lambda, v)$ be a labeled trace set. Let $\Gamma_{L,LTS}$ be the set of all legal alignment moves. A *cost function* $\kappa : \Gamma_{L,LTS} \rightarrow \mathbb{N}$ assigns a non-negative cost to each legal move. The overall *cost of a sequence of alignment moves* $\gamma \in \Gamma_{L,LTS}^*$ is computed as the sum of the cost of all constituent moves:

$$K(\gamma) = \sum_{(e,s) \in \gamma} \kappa(e,s).$$

◇

This cost function can be used to favor one type of explanation for deviations over the other. The cost of each legal move depends on the specific model and process domain and, hence, the cost function κ needs to be defined specifically for each setting.

Given a user-defined cost function, we define an *optimal alignment* that minimizes the deviations between log trace and process trace. Deviations are scored according to a user-defined cost function so that domain knowledge on the reliability of events and process activities can be used. An optimal alignment according to the user-defined cost function can be seen as the most likely mapping between events and process steps. Note that an optimal alignment does not need to be unique, i.e., multiple complete alignments with the same minimal cost may exist.

Definition 4.4 (Optimal alignment). An alignment $\gamma_\sigma \in \Gamma_{L,LTS}^\sigma$ is an *optimal alignment* for a given trace set LTS and log trace σ if and only if any other alignment between the log trace and the labeled trace set has the same or higher cost:

$$\forall \bar{\gamma}_\sigma \in \Gamma_{L,LTS}^\sigma (\bar{\gamma}_\sigma \text{ is an alignment} \wedge K(\gamma_\sigma) \leq K(\bar{\gamma}_\sigma)). \quad \diamond$$

4.3.3 Choice of a Cost Function

The choice of a good cost function is challenging. As illustrated in Example 4.1, there is no universally good cost function. Often, the cost function can be designed based on domain knowledge on the likelihood of deviations. It may be, e.g., based on knowledge on the reliability of event logging or on the likelihood of deviations. Another possibility to determine a good cost function is by comparing the resulting alignments to a *gold standard* alignment determined by process stakeholders. In case of missing information or as a starting point, the following *standard cost function* can be used.

Definition 4.5 (Standard cost function). The standard cost function is defined as $\kappa_1 : \Gamma_{L,LTS} \rightarrow \mathbb{N}$ with:

$$\kappa_1((e, (t, w))) = \begin{cases} 1 & \text{if } (e, (t, w)) \text{ is a log move} \\ 1 + |(dom(w))| & \text{if } (e, (t, w)) \text{ is a model move} \\ | \{v \in dom(w) \mid w(v) \neq \#_{v(v)}(e)\} | & \text{if } (e, (t, w)) \text{ and } \lambda(t) \neq \tau \\ 0 & \text{if } (e, (t, w)) \text{ is an incorrect synchronous move} \\ & \text{otherwise.} \end{cases}$$

\diamond

This standard cost function assigns the same cost of 1 to each deviation. Moreover, for each incorrect or missing variable assignment a cost of 1 is added. The only exceptions are model moves for unobservable transitions (i.e., mapped to τ). These model moves are assigned a cost of 0 since executions of those transitions can never be recorded in an event log.

Example 4.2 (Standard cost function). Using the standard cost function, we obtain the following cost for the alignments shown in Table 4.1. The cost of the alignment γ_{σ_2} in Table 4.1a is $K_1(\gamma_{\sigma_2}) = 3$. There is one incorrect synchronous move with one incorrect variable time_{che} (cost 1), two model moves of which one model move is for an invisible routing transition (cost 1), and one log move (cost 1). The cost of the alignment γ_{σ_4} in Table 4.1b is $K_1(\gamma_{\sigma_4}) = 2$ since there is one incorrect synchronous move with one incorrect variable (cost 1), one log move (cost 1) and one model move for an invisible routing transition (cost 0). The cost of the alternative alignment for trace σ_4 in Table 4.1c is $K_1(\bar{\gamma}_{\sigma_4}) = 8$ since there are five model moves for visible transitions (each cost 1) and three missing variable assignments (each cost 1). For the example cost function, γ_{σ_2} and γ_{σ_4} are optimal alignments. No other alignment with lower cost exists. Alignment $\bar{\gamma}_{\sigma_4}$ is clearly not optimal when using the standard cost function since alignment γ_{σ_4} has lower cost.

We introduced a formal notation for alignments, which provides a mapping between the events recorded in a log trace and the behavior defined by a process model. The next section concludes the introduction to multi-perspective conformance checking by showing how to use alignments to quantify the fitness quality dimension.

4.4 Measuring Fitness Based on Alignments

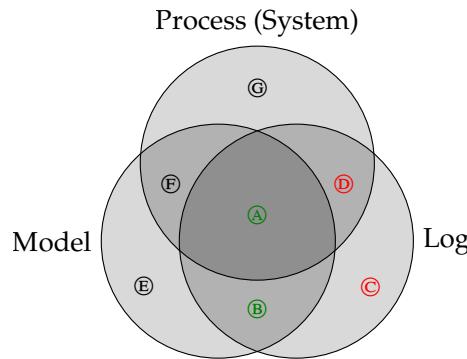


Figure 4.6: Venn diagram illustrating the fitness measure [BDA14].

When focusing on the fitness dimension of conformance, we are not only interested in finding the optimal alignment and, hence, diagnosing where a log trace does not conform to a model. Also, we wish to quantify the fitness level of traces $\sigma \in \mathcal{E}$

and logs $L = (E, \Sigma, \#, \mathcal{E})$ with regard to the process model LTS. We want to measure how much of the behavior that was recorded in the event log $(\textcircled{A} + \textcircled{B} + \textcircled{C} + \textcircled{D})$ can be explained by the process model, i. e., we want to determine the fraction: $\frac{\textcircled{A} + \textcircled{B}}{\textcircled{A} + \textcircled{B} + \textcircled{C} + \textcircled{D}}$. A process model is fitting well the event log when the area $\textcircled{C} + \textcircled{D}$ in Figure 4.6 is rather small.

For this reason, we introduce a *fitness function* for a log trace and a process model: $fitness(\sigma, LTS) \in [0, 1]$:

- $fitness(\sigma, LTS) = 1$, if the trace σ can be replayed by the model from the beginning to the end with no discrepancies; conversely,
- $fitness(\sigma, LTS) = 0$ denotes the poorest level of conformance.

We use the cost of a deviation according to the cost function used to determine the alignment as indication of its severity. Note that it would also be possible to use a second cost function as proposed by Adriansyah et al. [Adr14] in the case of control-flow alignments. The second cost function enables to assign different cost to the severity and likelihood of deviations. Here, we simplify the presentation by using the same cost function for both tasks.

However, the cost of all deviations $K(\sigma)$ cannot directly be used as fitness function as we want to obtain a fitness level between 0 and 1. Normalization can be done in multiple ways. We divide the cost of an optimal alignment by a reference cost, which is obtained using a worst case alignment that is always possible. Therefore, the *fitness level* of a log trace is defined with respect to a worst case scenario. The following fitness measure is an extended version of the *unbalanced fitness measure* presented in [LA13a]. In [LA13a], deviations on the control-flow perspective were treated separately from deviations on other perspectives.

Definition 4.6 (Fitness measure). Let $\sigma = \langle e_1, \dots, e_n \rangle \in \mathcal{E}$ be a log trace of event log $L = (E, \Sigma, \#, \mathcal{E})$. Let LTS be a trace set. Let $\gamma_{\sigma}^{\text{opt}} \in \Gamma_{L, \text{LTS}}^{\sigma}$ be an optimal alignment of σ and LTS. Let $\gamma_{\langle \rangle} \in \Gamma_{L, \text{LTS}}^{\langle \rangle}$ be an optimal alignment of the empty trace $\langle \rangle$ and LTS. Let $\gamma_{\sigma}^{\text{ref}} \in \Gamma_{L, \text{LTS}}^{\sigma}$ be a reference alignment given by

$$\gamma_{\sigma}^{\text{ref}} = \gamma_{\langle \rangle} \oplus \langle (e_1, \gg), \dots, (e_n, \gg) \rangle.$$

The fitness measure of σ and LTS is defined as:

$$fitness(\sigma, LTS) = 1 - \frac{K(\gamma_{\sigma}^{\text{opt}})}{K(\gamma_{\sigma}^{\text{ref}})}$$

◇

To compute the fitness, the cost of the optimal alignment is confronted with the cost of the reference alignment $K(\gamma_{\sigma}^{\text{ref}})$. The reference alignment is build by concatenating log moves for all events of σ and the alignment of the empty trace. $\gamma_{\sigma}^{\text{ref}}$ is used as the reference alignment as it contains no correct synchronous moves, which is an undesirable worst case scenario. Since $\gamma_{\sigma}^{\text{opt}}$ is an optimal alignment its

cost is optimal by definition: $0 \leq K(\gamma_{\sigma}^{\text{opt}}) \leq K(\gamma_{\sigma}^{\text{ref}})$. The fitness measure is *properly normalized*, i.e.:

$$0 \leq \text{fitness}(\sigma, \text{LTS}) \leq 1.$$

Example 4.3 (Fitness measure). Take the alignments shown in Table 4.1a, alignment $\gamma_{\sigma_2} \in \Gamma_{\text{L,LTS}}^{\sigma_2}$ is an optimal alignments for the trace σ_2 . Assume that the reference alignment for the labeled trace set LTS based on the example DPN (Figure 3.4) is as follows:

```
((>, (ttri, datacolor ↦ Red)),  
(>, (treg, (timereg ↦ 00/00/00 00:00, resreg ↦ Anyone))),  
(>, (τ2, ∅), (e20, >), (e21, >), ..., (e29, >))
```

with $K(\gamma_{\sigma_2}^{\text{ref}}) = 5 + 10$. The optimal alignment of the empty trace yield cost 5 and the ten log moves in the optimal alignment γ_{σ_2} yield cost 10. Then, the fitness measure for σ_2 with $K(\gamma_{\sigma_2}) = 2$ is: $\text{fitness}(\sigma_2, \text{LTS}) = 1 - \frac{2}{15} = 1 - 0.133 = 0.867$.

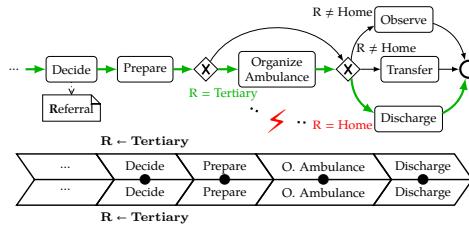
We introduced multi-perspective conformance checking and provided necessary preliminaries such as alignments and quantifying fitness based on alignments. In the next chapter, we describe a method to compute optimal alignments between a multi-perspective process model and an event log.

5 Multi-perspective Alignment

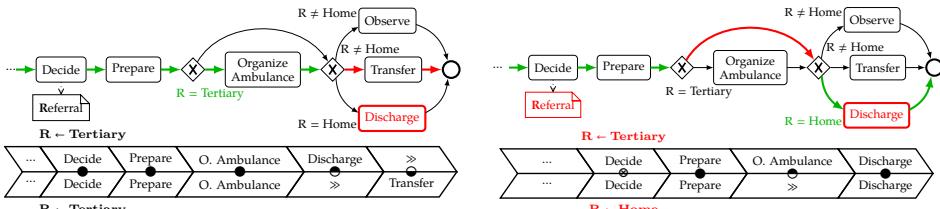
We present a method that returns an alignment between a multi-perspective process model and an event log. Our method allows to *balance deviations concerning the different perspectives* in a fully customizable manner.

5.1 Motivation for Balanced Alignments

We motivate the need for balanced, multi-perspective alignments using the three alignments illustrated in Figure 5.1.



(a) Invalid alignment when considering all process perspectives. According to the decision rules it is impossible that both *Organize Ambulance* and *Discharge* occurred.



(b) A valid alignment: Event *Discharge* is marked as *log move*, activity *Transfer* is introduced as *model move*, and the *referral* value is marked as correct.

(c) Another valid alignment: Event *Organize Ambulance* is marked as *log move* and the value recorded for the variable *referral* is diagnosed to be incorrect.

Figure 5.1: Motivation for a multi-perspective, balanced alignment. Three alternative alignments for log trace γ_{σ_2} (cf., Table 4.1a) illustrated using a fragment of the BPMN model of the hospital process.

Take the optimal alignment γ_{σ_2} introduced in Table 4.1a on page 61. When consid-

ering only the control-flow perspective, the execution of the two activities: *Organize Ambulance* and *Discharge* in a *single process instance* does not violate the behavior prescribed by the hospital process model. As illustrated in Figure 5.1a, both *Organize Ambulance* and *Discharge* would be correct synchronous moves in an alignment based on the control-flow perspective only. However, the decision rules of both activities contradict each other: There is no value for the variable *referral* that fulfills both rules ($\text{data}_{\text{referral}} = \text{Tertiary}$ and $\text{data}_{\text{referral}} = \text{Home}$). The alignment depicted in Figure 5.1a is invalid when considering all process perspectives.

Figures 5.1b and 5.1c illustrate two valid multi-perspective alignments. One possibility is that the recorded referral value was *Tertiary*. Then, *Organize Ambulance* should happen and *Discharge* must not happen as diagnosed by the alignment shown in Figure 5.1b. Another explanation is that the referral was *Home*. Then, *Discharge* can be executed, but *Organize Ambulance* must not occur as diagnosed by the alignment shown in Figure 5.1c. These examples illustrate that alignments need to balance between deviation in multiple perspectives.

Our proposed method balances the control-flow, data, resources, and time perspectives in identifying explanations for deviations. Depending on the importance of the perspectives either the alignment shown in Figure 5.1b or the alignment shown in Figure 5.1c can be considered the best explanation, i. e., an optimal alignment. The method is customizable through a user-defined cost function that encompasses all process perspectives. Users can assign different weights to the different types of deviations. In fact, there might be situations in which process attributes are known to be more reliable than the activities recorded as events. This can be reflected in the weights by assigning a higher cost to deviations on the data perspective. In such situation, the alignment in Figure 5.1c would be a better explanation for the log trace than Figure 5.1c. In Figure 5.1c the value recorded for *referral* is assumed to be correct.

The remainder of this chapter is divided in four sections: In Section 5.2, we present the alignment method; in Section 5.3, we show that our approach is feasible to be applied on real-life event logs and process models; in Section 5.4, we discuss related work in the fields of process mining; and in Section 5.5 we conclude by summarizing the contributions and sketch future work.

5.2 Balanced Alignment Method

In this section, we present our balanced alignment method for multi-perspective process models, which has been published in [Man+14; Man+16a]. Before presenting the algorithm, we elaborate on the required input.

As outlined in Figure 5.2, the input to the balanced alignment method is a process model in form of a DPN, an event log, and a user-defined cost function. The cost function can be used to provide domain knowledge on the likelihood of deviations and is used to guide the search for an optimal balanced alignment. We require

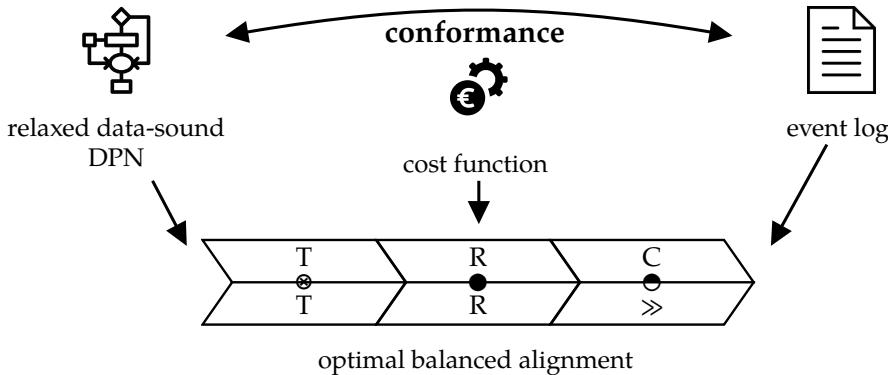


Figure 5.2: Overview of the proposed balanced alignment method.

the process models to be DPNs since the trace set notation is unable to provide procedural execution semantics, which are required to explore the possibly infinite set of process traces. As discussed in Section 3.2, DPNs have clear semantics that makes them suitable to describe multi-perspective process models used in practice. However, as we defined alignments (cf., Definition 4.2) based on trace sets, we show that the result of our method is a valid alignment for the trace when converting the language of the DPN to a trace sets.

5.2.1 Assumptions on the Input

Two important assumptions are made with regard to the input:

- The DPN should be **relaxed data sound** and
- DPN guards should be **linear guard expressions**.

In a relaxed data-sound DPN there needs to be at least *one sequence of transition firings* that leads from the initial state to a final state. This restriction is motivated by the fact that the process projection of an alignment is required to be a process trace of the model. Clearly, to be able to provide an alignment, the DPN should allow for at least one trace.

Definition 5.1 (Relaxed data-sound DPN). Let $N = (P, T, F, V_P, \text{dom}, \text{in}, \text{wr}, \text{gd})$ be a DPN with initial marking M_I and final marking M_F . DPN N is *relaxed data sound* if and only if its trace set contains at least one complete trace: $\text{TS}_{N, M_I, M_F} \neq \emptyset$, i.e., there exists one firing sequence from the initial marking to one of the the final states in the final marking. \diamond

Relaxed data-sound DPNs are *not required* to be free from deadlocks, i.e., states different from one of the final state in which no transitions are enabled, or livelocks,

i. e., firing sequences that do not progress towards a final state. However, we require that one of the final states is reachable from the initial marking. The requirement that the DPNs is relaxed sound does not hinder the applicability of our method in practice. In fact, a business process models that does not define at least one possible process trace can be discarded for analysis anyway.

Furthermore, our method requires guards to be linear boolean expressions as defined in Definition 2.8. It would be impossible to support arbitrary guard expressions since we need to be able to find variable assignments that satisfy the guard expressions. However, satisfiability of non-linear constraints over integers is undecidable [DH73].

5.2.2 A* Algorithm and Search Space

We formulate the problem of finding an optimal alignment between a multi-perspective process model and a log trace as a search problem in a directed graph and employ the A* algorithm [DP85] to find a least expensive path through the graph. We briefly introduce the A* algorithm.

Let $Z = (Z_V, Z_E)$ be a directed graph with edges weighted based on a predefined cost structure. The A* algorithm, as initially proposed in [DP85], finds the path with the overall lowest cost from a given source node $v_0 \in Z_V$ to a node of a given goal set, i. e. a set of target nodes $Z_G \subseteq Z_V$. Each node $v \in Z_V$ is associated with a cost that is determined by an *evaluation function* $f(v) = g(v) + h(v)$, where

- $g : Z_V \rightarrow \mathbb{R}^+$ gives the *smallest path cost* from v_0 to v ;
- $h : Z_V \rightarrow \mathbb{R}_0^+$ gives an *estimate of the smallest path cost* from v to any goal node $v_G \in Z_G$ from v .

Function h is *admissible* if it always *underestimates the remaining cost* to reach any goal node v_G from v : for each node $v \in Z_V$ and for each goal node $v_G \in Z_G$ that is reachable from v , $h(v) \leq g(v_G) - g(v)$ holds. If h is an admissible function, then A* is *guaranteed* to return a path that has *lowest overall cost*.

In order to use A* to find an optimal alignment, the search space needs to be defined along with the cost of paths in the search-space.

Definition 5.2 (Alignment search space). Let $LN = (N, \lambda, \nu)$ be a labeled DPN with initial marking M_I and final marking M_F . Let $LTS = (TS_{N, M_I, M_F}, \lambda, \nu)$ be the trace set of DPN N having the same labeling functions λ and ν . Let σ be a log trace of the event log $L = (E, \Sigma, \#, \mathcal{E})$, then, the *search space* to find an optimal alignment of LN and σ is a graph $Z_{LN, M_I, M_F, \sigma} = (Z_V, Z_E)$ where:

- the set of nodes Z_V contains sequences of alignment moves s. t. their process projection corresponds to a process trace in the DPN (not necessarily complete) and their log projection is a prefix of the log trace:

$$Z_V = \{\gamma \in \Gamma_{L, LTS}^* \mid proj_L(\gamma) \in prefix(\sigma) \wedge proj_P(\gamma) \in TS_{N, M_I}\};$$

- the set of edges Z_E contains all $(\gamma_\sigma, \bar{\gamma}_\sigma) \in Z_V \times Z_V$, where $\bar{\gamma}_\sigma$ is obtained by adding one legal alignment move to γ_σ :

$$Z_E = \{(\gamma, \bar{\gamma}) \in Z_V \times Z_V \mid \exists_{(e,s) \in \Gamma_{L,LTS}} (\bar{\gamma} = \gamma \cdot (e,s))\}.$$

The set of goal nodes $Z_G \subseteq Z_V$ contains all alignments of σ and the trace set TS_{N,M_I,M_F} of the DPN N :

$$Z_G = \{\gamma_\sigma \in \Gamma_{L,LTS}^* \mid proj_L(\gamma_\sigma) = \sigma \wedge proj_P(\gamma_\sigma) \in TS_{N,M_I,M_F}\} = \Gamma_{L,LTS}^\sigma. \quad \diamond$$

The search space Z consists of sequences of alignment moves, some of these sequences are prefixes of an alignment. Therefore, we use γ to denote nodes in Z . To find an optimal alignment of N and σ , we search a path in Z with the lowest cost from the source node $\gamma_0 = \langle \rangle$ to a goal node $\gamma_\sigma \in Z_G$.

Definition 5.3 (Cost of a path in the search space). Let $Z_{LN,M_I,M_F,\sigma} = (Z_V, Z_E)$ be an alignment search space. We define the *cost* associated with a path leading to a graph node $\gamma \in Z_V$ as:

$$g(\gamma) = K(\gamma) + \epsilon |\gamma|$$

Cost epsilon is a sufficiently-small negligible cost $\epsilon \in \mathbb{R}^+$, which we use to guarantee termination of the search algorithm (see Theorem 5.2). Adding cost ϵ does not affect the optimality of the returned alignment as long as it is chosen sufficiently small. \diamond

The A* algorithm guarantees to find such a path only if the cost is monotonically increasing while more nodes are added to the path. The following theorem proves that the cost from Definition 5.2 satisfies a stricter form of this property.

Theorem 5.1 (Cost g is strictly increasing). Let $Z_{LN,M_I,M_F,\sigma} = (Z_V, Z_E)$ be an alignment search space. Let $\gamma, \bar{\gamma} \in Z_V$ be two nodes in the alignment search space such that $(\gamma, \bar{\gamma}) \in Z_E$, i. e., there is an edge from γ to $\bar{\gamma}$. Let $g(\gamma) = K(\gamma) + \epsilon |\gamma|$ be the cost associated with a path leading to a graph node $\gamma \in Z_V$. Then $\forall_{\bar{\gamma} \in Z_V} (g(\gamma) < g(\bar{\gamma}))$. \diamond

PROOF. Assume that there is an alignment $\bar{\gamma} \in Z_V$ with lower cost than γ : $g(\bar{\gamma}) < g(\gamma)$. As $(\gamma, \bar{\gamma}) \in Z_E$, it follows that there exists $(e, s) \in \Gamma_{L,LTS}$ s. t. $\bar{\gamma} = \gamma \oplus (e, s)$. The cost of a path to a node in Z_V is defined as the sum of the cost of all moves that led to the node: $g(\bar{\gamma}) = K(\gamma) + \epsilon |\gamma| = \sum_{(e,s) \in \gamma} \kappa(e,s) + \epsilon |\gamma|$. Therefore, the cost of $\bar{\gamma}$ can be expressed as $g(\bar{\gamma}) = g(\gamma) + \kappa((e,s)) + \epsilon$. Using the assumption $g(\bar{\gamma}) = g(\gamma) + \kappa((e,s)) + \epsilon < g(\gamma) \Leftrightarrow \kappa((e,s)) + \epsilon < 0$ should hold, but $\kappa \in \Gamma_{L,LTS} \rightarrow \mathbb{R}_0^+$ is non-negative and $\epsilon \in \mathbb{R}^+$ is positive. \square

Regarding the heuristic function h , there are multiple options available.

1. We set $h(\gamma) = 0$ and, thus, naïvely underestimating the remaining cost. This turns the A* search algorithm into the uninformed Dijkstra's algorithm [Dij59], in which there is no pruning of the search space.

2. We use the heuristic function introduced by Adriansyah et al. [Adr14], which uses the Petri net marking equation to estimate the remaining cost.

We assume that option (2), i. e., the marking equation heuristic is employed. We defer a more detailed discussion of this heuristic and its applicability to the multi-perspective alignment problem to Section 5.2.7.

5.2.3 Searching an Optimal Balanced Alignment

Defining the search space and an admissible path cost function would be sufficient to obtain an optimal solution by direct application of the A* algorithm. However, there are two problems regarding our definition of the search space in Definition 5.2 that we are solving in this section:

1. the set of nodes Z_V may be infinite,
2. the set of edges $(\gamma, \bar{\gamma}) \in Z_E$ associated to a single node γ may be infinite.

Thus, a direct application of A* might require building an infinite search space up-front. Moreover, an infinite number of successor nodes may need to be explored in a single iteration of the search algorithm. Clearly, this is infeasible. Therefore, before introducing the actual algorithm, which iteratively builds only part of the search space, we introduce the concepts of *control-flow successors* and the *augmentation with variable assignments*.

Exploring the Control-flow Successors

The idea is to explore successors of a node in the search space based on the control-flow perspective, and, then, augment with the other perspectives. Figure 5.3 illustrates this idea. The potentially infinite number of successors of a node in the search space Z_V is reduced to a small number of successors when taking only the control-flow perspective into account.

We define two necessary notations to formally introduce the set of control-flow successors of a node in the search space. First, we define the projection function $proj_{CF} : \Gamma_{L,LTS}^* \rightarrow \Gamma_{L,LTS}^*$ that removes all variable assignments from a sequence of alignment moves $\gamma \in \Gamma_{L,LTS}^*$. We define $proj_{CF}(\gamma)$ recursively:

$$\begin{aligned} proj_{CF}(\langle \rangle) &= \langle \rangle \\ proj_{CF}(\langle (e, \gg) \rangle \cdot \gamma) &= \langle (e, \gg) \rangle \cdot proj_{CF}(\gamma) \\ proj_{CF}(\langle (\gg, (t, w)) \rangle \cdot \gamma) &= \langle (\gg, (t, \emptyset)) \rangle \cdot proj_{CF}(\gamma) \\ \forall_{e \neq \gg} (proj_{CF}(\langle (e, (t, w)) \rangle \cdot \gamma) &= \langle (e, (t, \emptyset)) \rangle \cdot proj_{CF}(\gamma)). \end{aligned}$$

Second, we introduce the control-flow copy of a DPN. We denote a copy of the original DPN N in which in all variables, write operations, and guards are removed as control-flow copy of N.

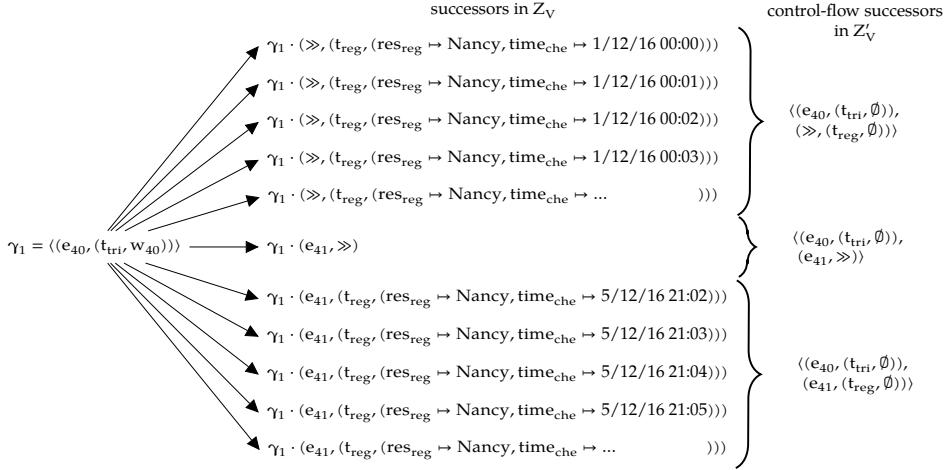


Figure 5.3: Collapsing the search space by considering only the control-flow successors of a node. A potentially infinite number of successors of γ_1 in Z_V is reduced to exactly three successors in Z'_V : one model move, one log move, and one correct synchronous move.

Definition 5.4 (Control-flow copy of a DPN). Let $LN = (N, \lambda, \nu)$ be a labeled DPN. We define the *control-flow copy* of DPN LN as $LN' = (N', \lambda, \emptyset)$ with:

$$\begin{aligned} N' &= (P, T, F, \emptyset, \emptyset, wr', gd'), \text{ s.t.,} \\ \forall_{t \in T} (wr'(t) = \emptyset \wedge gd'(t) = \text{true}). &\quad \diamond \end{aligned}$$

We define the set of control-flow successors of a node in the search space, i. e., a function that returns only the nodes depicted on the right side in Figure 5.3.

Definition 5.5 (Control-flow successors). Let $Z_{LN, M_I, M_F, \sigma} = (Z_V, Z_E)$ be an alignment search space. Let $Z'_{LN', M_I, M_F, \sigma} = (Z'_V, Z'_E)$ be the alignment search space based on the control-flow copy DPN LN' . We denote the set of *control-flow successors* of a node $\gamma \in Z_V$ with

$$ctrlSuccessors_{LN, M_I, \sigma}(\gamma) \subseteq Z'_V.$$

We obtain $ctrlSuccessors_{LN, M_I, \sigma}(\gamma)$ by concatenating γ with one legal alignment move (e, s) such that the added process step s corresponds to either \gg or a transition firing in LN' . Also, we remove the variable assignments from all moves:

$$ctrlSuccessors_{LN, M_I, \sigma}(\gamma) = \{\gamma_C \in Z'_V \mid \exists_{(e, s) \in \Gamma_{L, LTS}} (\gamma_C = proj_{CF}(\gamma) \cdot (e, s))\}, \quad \diamond$$

Note a control-flow successor $\gamma_C \in Z'_V$ is not necessarily part of the search space Z_V , since $Z_V \neq Z'_V$. All variable assignments are missing in the process projection of γ_C .

Augmenting with Optimal Variable Assignments

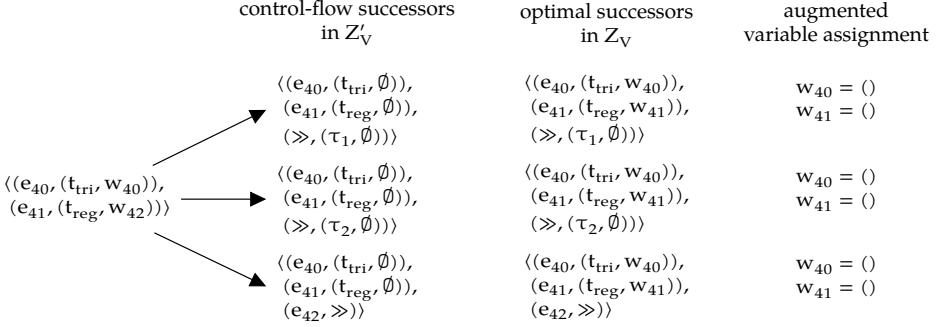


Figure 5.4: Augmentation with variable assignments of a control-flow successor.

As indicated, not every control-flow successor γ_C is a node of the search space. Control-flow successors γ_C need to be augmented with the write operations. Variables can be defined on infinite domains and, as a result, γ_C may have an infinite number of possible augmentations. Since we aim to minimize the alignment cost, we only take one of these augmentations: the augmentation with the lowest cost. Figure 5.4 illustrates this augmentation and the collapsing of a, potentially, infinite number of successors to a small finite number of successor nodes.

We perform the augmentation of the variable assignments in a similar way as discussed in [LA13a] with the notable difference that we also augment alignments of prefixes of the log trace with the process model. This is required to obtain an optimal alignment that is balanced according to all process perspectives. By contrast, in [LA13a] only alignments of entire log traces are augmented, which may result in suboptimal or invalid alignments being returned. For example, the invalid alignment illustrated in Figure 5.1a would be returned.

Definition 5.6 (Augmentation function). Let $Z_{LN, M_I, M_F, \sigma} = (Z_V, Z_E)$ be an alignment search space. Let $Z'_{LN', M_I, M_F, \sigma} = (Z'_V, Z'_E)$ be the alignment search space based on the control-flow copy DPN LN' . Let $AUG_{\gamma_C} = \{\gamma_D \in Z_V \mid \forall \bar{\gamma} \in Z_V (proj_{CF}(\gamma_D) = \gamma_C \wedge ((proj_{CF}(\gamma_D) = proj_{CF}(\bar{\gamma})) \implies (g(\gamma_D) \leq g(\bar{\gamma})))\}$ be the set of possible variable augmentations for a control-flow successor $\gamma_C \in ctrlSuccessors_{LN', M_I, \sigma}(\gamma)$ with lowest cost, i. e., *optimal variable augmentations*. The *variable augmentation function*:

$$augmentVariables_{LN, \sigma, K} : Z'_V \rightarrow (AUG_{\gamma_C} \cup \{\top\}),$$

returns one of the possible augmentations of node γ_C from the set of possible augmentations AUG_{γ_C} or \top if no such augmentation exist. \diamond

The implementation of the augmentation function is described separately in Section 5.2.5. In case there are multiple optimal variable augmentations for a control-

flow successor, the augmentation function may return any one of them. Moreover, it may happen that γ_C cannot be augmented with other perspectives, i. e., there is no variable assignment that fulfills the guards of all DPN transitions linked to the control-flow alignment.⁸ In this case, the function is assumed to return the special value T . Note that the augmentation γ_D needs to be computed from scratch for each for control-flow successor γ_C , ignoring the assignment computed for previous nodes. Indeed, the last move may refer to a transition t that is not allowed to fire in the DPN state reached after the process trace $\text{proj}_P(\gamma)$.

Balanced Alignment Algorithm

Input: Labeled DPN (LN), Initial and final markings (M_I, M_F), Log trace (σ), Cost function (K)
Result: Balanced alignment (γ)

```

1  $\gamma \leftarrow \langle \rangle$ 
2 QUEUE =  $\langle \rangle$ 
3 while  $\neg(\gamma \text{ is an alignment})$  do
4   foreach  $\gamma_C$  in  $\text{ctrlSuccessors}_{LN, M_I, \sigma}(\gamma)$  do
5      $\gamma_D \leftarrow \text{augmentVariables}_{LN, \sigma, K}(\gamma_C)$ 
6     if  $\gamma_D \neq T$  then
7        $g(\gamma_D) \leftarrow K(\gamma_D) + \epsilon |\gamma_D|$ 
8        $f(\gamma_D) \leftarrow g(\gamma_D) + h(\gamma_D)$ 
9       enqueue(QUEUE,  $\gamma_D, f(\gamma_D))$ 
10     $\gamma \leftarrow \text{pollLowestCost}(QUEUE)$ 
11 return  $\gamma$ 
```

Algorithm 1: Procedure that computes a balanced alignment

Algorithm 1 illustrates how we use the A* algorithm to search for an optimal alignment. The algorithm takes a relaxed data-sound, labeled DPN LN and a log trace σ as input and returns the optimal alignment γ_σ that is balanced according to a given cost function K. Instead of building the graph Z beforehand – which is potentially infinite – we build up the search space incrementally.

Starting with the empty sequence $\gamma = \langle \rangle$ as its source node (line 1), we build the set $\text{ctrlSuccessors}_{LN, M_I, \sigma}(\gamma)$ with all successors of γ by taking only the control-flow perspective into account (line 3). By doing so, we prune the alignment search space $Z_{LN, M_I, M_F, \sigma}$ such that there is only a finite number of edges connected to any given node $\gamma \in Z_V$. The possibly infinite number of variable assignments for any given legal alignment move is reduced to exactly *one variable assignment*, which results in a minimal cost for the next node in the search space γ , i. e., an optimal variable

⁸There may be deadlocks based on the data constraints defined in the DPN.

assignment. It is not important which one of the optimal variable assignments is chosen as long as it results in the minimal cost.

We use function $\text{augmentVariables}_{\text{LN}, \sigma, K}$ to augment the control-flow successors with a variable assignment (line 5). If an augmentation γ_D does not exist, γ_C does not yield to any valid alignment to be added to QUEUE. Otherwise, if an augmentation γ_D exists, i.e., $\gamma_D \neq \top$, then, the cost $f(\gamma_D)$ is computed (line 7-8). Thereafter, we inserted node γ_D into the priority queue QUEUE using the function enqueue (line 9).

Once all the control-flow successors are considered, a new node γ is picked from the head of QUEUE using the function pollLowestCost , i.e., one of the nodes associated with the lowest cost (line 10s). If the node γ is an alignment according to Definition 4.2 (i.e., $\gamma \in Z_G$), then it is returned as the optimal alignment (line 3 and line 11). Since the heuristic function h is admissible and the cost g is monotonically increasing, the application of A* guarantees that the first goal node visited has the lowest cost of all goal nodes. Otherwise, all successors are processed as described beforehand (line 4-10). Therefore, the search-space successors of a given node are only created when such a node is visited, without unnecessarily storing information for search-space nodes that are never going to be visited.

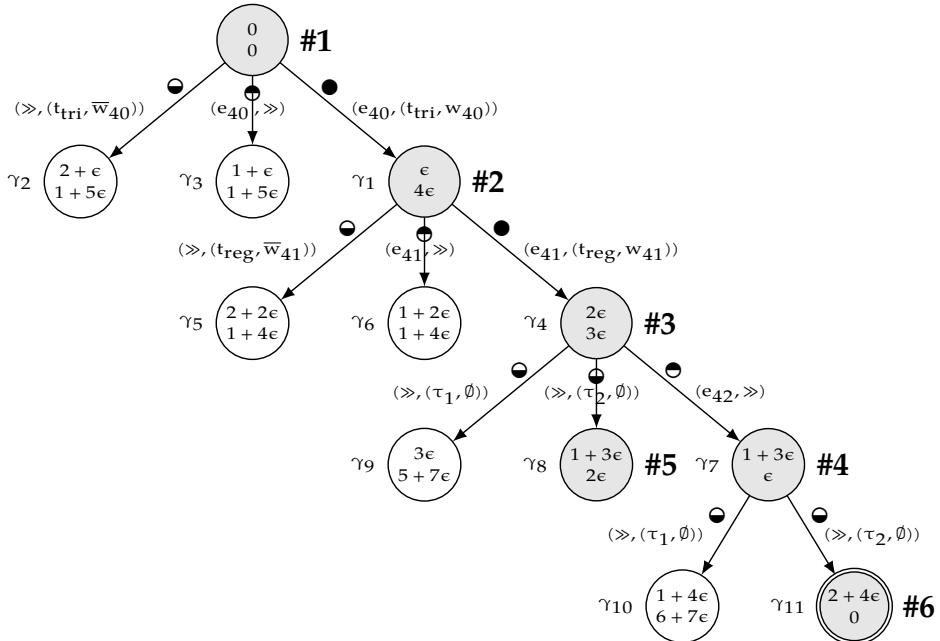


Figure 5.5: Search space explored for an optimal alignment of σ_4 and the hospital process.

Example 5.1 (Incremental Building of the Search Space). Figure 5.5 shows the portion of the search space that Algorithm 1 constructs to find an optimal alignment of σ_3 and the hospital example DPN using the standard cost function K_1 . Nodes $\gamma \in Z_V$ in the search space are represented by circles. Goal nodes $\gamma_G \in Z_G$ are depicted with a double-line border. Nodes that have been polled from the priority queue during the search are emphasized with a gray background. The remaining nodes have been enqueued in the priority queue, but have never been polled since their cost exceeds the cost of an optimal alignment. Polled nodes are assigned numbers #1, ..., #6 that indicate the order in which they have been visited. We include both the values for the actual cost $g(\gamma)$ (first value) and the estimated cost to reach a goal node $h(\gamma)$ (second value). Edges between two nodes γ and $\bar{\gamma}$ are labeled with the type of alignment move (i. e., synchronous move \bullet , incorrect synchronous move \otimes , model move \ominus , and log move \oplus) and the alignment move (e, s) with which γ has been extended. For example, $\gamma_2 = \langle \rangle \cdot ((e_{40}, (t_{tri}, w_{40})))$.

Algorithm 1 starts with the empty alignment $\langle \rangle$ (#1). Both functions g and h are initially 0. Then, three control-flow successors are generated by $ctrlSuccessors_{LN, M_1, \sigma}$ (Algorithm 1, line 4):

$$\gamma_C^1 = \langle (\gg, (t_{tri}, \emptyset)) \rangle, \quad \gamma_C^2 = \langle (e_{40}, \gg) \rangle, \quad \gamma_C^3 = \langle (e_{40}, (t_{tri}, \emptyset)) \rangle.$$

In γ_C^1 a model move for process transition t_{tri} was appended, in γ_C^2 a log move for event e_{30} was appended, and in γ_C^3 a synchronous move for transition t_{tri} and event e_{30} . An optimal *variable assignment* according to the guard expressions of the DPN is determined for γ_C^1 , γ_C^2 and γ_C^3 using function $augmentVariables_{LN, \sigma, K}$ (Algorithm 1, line 7). The obtained variable assignment may differ from the attribute values observed in the event in order to fulfill guards specified in the DPN. Thus, synchronous moves (\bullet) might be turned into incorrect synchronous moves (\otimes). In Figure 5.5, we obtain nodes:

$$\begin{aligned}\gamma_1 &= \langle (\gg, (t_{tri}, \bar{w}_{40} = (\text{data}_{\text{color}} \mapsto \text{White}))) \rangle \\ \gamma_2 &= \langle (e_{40}, \gg) \rangle \\ \gamma_3 &= \langle (e_{40}, (t_{tri}, w_{40} = (\text{data}_{\text{color}} \mapsto \text{Green}))) \rangle.\end{aligned}$$

The assignment $\bar{w}_{40} = (\text{data}_{\text{color}} \mapsto \text{White})$ chosen for the model move in γ_1 is correct because γ_1 does not contain any move that is associated to a guarded transition. Therefore, variable $\text{data}_{\text{color}}$ is free to take on any value in its domain. The assignment $w_{40} = (\text{data}_{\text{color}} \mapsto \text{White})$ chosen for the synchronous move in γ_3 is correct since function $augmentVariables_{LN, \sigma, K}$ returns the attribute value of event e_{40} if it does not violate any guards. Because there is not yet any guard constraining the value of variable $\text{data}_{\text{color}}$, the observed value can be considered as correct. We elaborate further the computation of the optimal variable assignments in Section 5.2.5.

After augmentation with optimal variable assignments, the path cost for all three generated sequences is calculated and they are enqueued into the priority

queue QUEUE (Algorithm 1, lines 8-12). Based on the standard cost function K_1 , we obtain the following cost:

$$g(\gamma_1) = \epsilon, \quad g(\gamma_2) = 2 + \epsilon, \quad g(\gamma_3) = 1 + \epsilon$$

Each deviating move contributes cost 1. The values returned by the heuristic function are based on the minimum number of model/log moves that need to be executed to reach a goal node. Further information is given in Section 5.2.7. Then, the sequence of alignment moves with the lowest overall cost is polled from the queue (Algorithm 1, line 13). Node γ_1 has the lowest cost $f(\gamma_1) = g(\gamma_1) + h(\gamma_1) = \epsilon + 4\epsilon$ among all nodes in the priority queue.

Again, control-flow successors are generated based on γ_1 and further nodes are visited in the same manner until a goal node is encountered. The goal node first visited is an optimal alignment between σ and LN and returned (Algorithm 1, line 12) and returned. Here, γ_{11} is an optimal alignment. Transition τ_2 is preferred over transition τ_1 despite the violated guard since it leads to the shorter path to a final marking.

5.2.4 Formal Guarantees

In the following two sections we show that Algorithm 1 terminates and returns an optimal alignment for any given log trace and relaxed data-sound DPN.

Algorithm 1 terminates

We proof that Algorithm 1 terminates. As necessary preliminary for the proof, we show that it is possible to find an variable assignment for any prefix of a goal node $\gamma_\sigma \in Z_G$ using function $\text{augmentVariables}_{LN, \sigma, K}$. Intuitively, this property holds because the sequence of variable assignments of any prefix of a goal node needs to fulfill less or equal constraints as the sequence of variable assignment of the goal node.

Lemma 5.1 (Sequence of variable assignments exists prefixes of goal nodes). Let $Z_{LN, M_I, M_F, \sigma} = (Z_V, Z_E)$ be an alignment search space with $\sigma = \langle e_1, \dots, e_n \rangle$ and $LN = (N, \lambda, \nu)$. Let K be a cost function. Let $\gamma_\sigma \in Z_G$ be a goal node. Function $\text{augmentVariables}_{LN, \sigma, K}$ returns nodes in the search space when applied to the control-flow projection of any prefix of γ_σ :

$$\forall_{\gamma \cdot (e, s) \in \text{prefix}(\gamma_\sigma)} (\text{augmentVariables}_{LN, \sigma, K}(\gamma, \text{proj}_{CF}(\gamma \cdot (e, s))) \neq \top). \quad \diamond$$

PROOF. Let $\langle (e_1, (t_1, w_1)), \dots, (e_n, (t_n, w_n)) \rangle = \gamma_\sigma$. Assume that there exists a node $\gamma \cdot (e, s) = \langle (e_1, (t_1, w_1)), \dots, (e_m, (t_m, w_m)) \rangle \in \text{prefix}(\gamma_\sigma)$ with $m \leq n$ and with the control-flow projection $\gamma_C = \text{proj}_{CF}(\gamma \cdot (e, s))$. Assume that $\text{augmentVariables}_{LN, \sigma, K}(\gamma, \text{proj}_{CF}(\gamma \cdot (e, s))) \neq \top$.

$\gamma_C = T$, i. e., it is impossible to find variable assignments for the process steps of γ_C that fulfill all the constraints of the DPN LN.

Since $\gamma_\sigma \in Z_G$ is a goal node there exists variable assignments $w_1, \dots, w_m, \dots, w_n$ s. t.:

$$\langle(t_1, w_1), \dots, (t_m, w_m), \dots, (t_n, w_n) \rangle \in TS_{N, M_I, M_F}$$

Since $\langle(t_1, w_1), \dots, (t_m, w_m), \dots, (t_n, w_n) \rangle$ is a process trace of the DPN, i. e., the variable assignments fulfill all the constraints of the DPN, the variable assignments of any prefix $\langle(t_1, w_1), \dots, (t_m, w_m) \rangle$ also fulfill all the constraints of the DPN. Thus, function $augmentVariables_{LN, \sigma, K}$ could have returned node:

$$\langle(e_1, (t_1, w_1)), \dots, (e_m, (t_m, w_m)) \rangle = \gamma \cdot (e, s)$$

which contradicts our assumption. \square

Using Lemma 5.1, we proof that Algorithm 1 terminates when applied to any log trace and relaxed data-sound DPN. The proof is divided in three parts: (1) we show that any search space contains at least one goal node, i. e., there exists an alignment between the log trace and the DPN; (2) we show that Algorithm 1 visits at least one goal node, i. e., even though it explores only the control-flow successors of nodes and augments them with only one optimal variable assignment, the alignment is eventually constructed; and (3) we show that this alignment is visited after exploring a finite number of nodes. Together these three properties proof that the algorithm terminates when applied on an DPN and a log trace with a given cost function.

Theorem 5.2 (Algorithm 1 terminates). Let $Z_{LN, M_I, M_F, \sigma} = (Z_V, Z_E)$ be an alignment search space with $\sigma = \langle e_1, \dots, e_n \rangle$. Let K be a cost function. Algorithm 1 terminates with inputs LN, M_I , M_F , σ , and K. \diamond

PROOF. We need to show three properties: (1) there exists at least one alignment, i. e., $\gamma_\sigma \in Z_G$, (2) Algorithm 1 eventually visits this alignment $\gamma_\sigma \in Z_G$, and (3) Algorithm 1 explores a finite number of nodes before visiting γ_σ .

(1) *Alignment exists.* Since we require LN to be relaxed data sound, there exists at least one complete process trace $\sigma_N = \langle(t_1, w_1), \dots, (t_m, w_m) \rangle \in TS_{N, M_I, M_F}$. Therefore, there exists at least alignment $\gamma_\sigma = \langle(e_1, \gg), \dots, (e_n, \gg), (\gg, (t_1, w_1)), \dots, (\gg, (t_m, w_m)) \rangle \in Z_G$, which belongs to the target nodes Z_G of the search space explored by Algorithm 1.

(2) *Alignment is visited.* Suppose that Algorithm 1 does not visit any $\gamma_\sigma \in Z_G$. It follows that there exists a sequence of alignment moves $\gamma \cdot (e, s) \in prefix(\gamma_\sigma)$ that

is not enqueued to the priority queue in Algorithm 1. There may be two reasons that $\gamma \cdot (e, s)$ is not enqueued into the priority queue:

$$\text{proj}_{\text{CF}}(\gamma \cdot (e, s)) \notin \text{ctrlSuccessors}_{\text{LN}, M_1, \sigma}(\gamma) \quad (5.1)$$

or

$$\text{augmentVariables}_{\text{LN}, \sigma, K}(\text{proj}_{\text{CF}}(\gamma \cdot (e, s))) \neq \gamma \cdot (e, s). \quad (5.2)$$

Thus, either the control-flow successor (Equation 5.1) is not generated or the variable assignment is not returned (Equation 5.2). We distinguish two cases regarding the added alignment move:

- (A) $(e, s) = (e_i, \gg)$ (log move) and
- (B) $(e, s) = (\gg, (t_i, w_i))$ (model move).

In case (A), the sequence of alignment moves is

$$\gamma \cdot (e_i, \gg) = \langle (e_1, \gg), \dots, (e_{i-1}, \gg) \rangle \cdot (e_i, \gg)$$

since we assumed that in alignment γ_σ all log moves precede the model moves. Moreover, because $\gamma \cdot (e_i, \gg)$ is a prefix of the goal node γ_σ , it follows that $\langle e_1, \dots, e_i \rangle \in \text{prefix}(\sigma)$ is a prefix of the log trace. Sequence $\gamma \cdot (e, \gg)$ consists only of log moves. Log moves do not require any variable assignment. Therefore, no variable assignments need to be determined (i. e., all process steps are \gg) and Equation 5.2 does not hold. Consequently, Equation 5.1, $\text{proj}_{\text{CF}}(\gamma) \cdot (e, \gg) \notin \text{ctrlSuccessors}_{\text{LN}, M_1, \sigma}(\gamma)$, needs to hold. Since function $\text{ctrlSuccessors}_{\text{LN}, M_1, \sigma}(\gamma)$ returns all control-flow successors of γ (cf., Definition 5.5), it follows that $\text{proj}_{\text{CF}}(\gamma) \cdot (e, \gg) \notin Z'_V$ should hold. Thus, $\text{proj}_{\text{CF}}(\gamma) \cdot (e, \gg)$ is not a control-flow successors of γ . However, its process projection is the empty sequence and its log projection is a prefix of the log trace σ :

$$\begin{aligned} \text{proj}_P(\text{proj}_{\text{CF}}(\gamma) \cdot (e, \gg)) &= \langle \rangle \in \text{TS}_{N', M_1} \text{ and} \\ \text{proj}_L(\text{proj}_{\text{CF}}(\gamma) \cdot (e, \gg)) &= \langle e_1, \dots, e_i \rangle \in \text{prefix}(\sigma). \end{aligned}$$

This implies that $\text{proj}_{\text{CF}}(\gamma) \cdot (e, \gg) \in Z'_V$ is a control-flow successor, which contradicts our assumptions.

In case (B), the sequence of alignment moves is:

$$\begin{aligned} \gamma \cdot (\gg, (t_i, w_i)) &= \langle (e_1, \gg), \dots, (e_n, \gg), \\ &\quad (\gg, (t_1, w_1)), \dots, (\gg, (t_{i-1}, w_{i-1})) \rangle \cdot (\gg, (t_i, w_i)). \end{aligned}$$

From Equation 5.1 and Definition 5.5 it follows that $\text{proj}_{\text{CF}}(\gamma) \cdot (\gg, (t_i, \emptyset)) \notin Z'_V$ should hold, i. e., the node is not a control-flow successor of γ . However, we assumed that $\gamma \cdot (\gg, (t_i, w_i)) \in \text{prefix}(\gamma_\sigma)$. Moreover, each sequence of transition

firings of the original DPN LN is also a transition firing sequence of the control-flow copy DPN LN'. Therefore, the following two equations hold:

$$\begin{aligned} \text{proj}_P(\text{proj}_{CF}(\gamma) \cdot (\gg, (t_i, \emptyset))) &= \langle (t_1, \emptyset), \dots, (t_i, \emptyset) \rangle \in TS_{N', M_i} \text{ and} \\ \text{proj}_L(\text{proj}_{CF}(\gamma) \cdot (\gg, (t_i, \emptyset))) &= \langle e_1, \dots, e_n \rangle = \sigma \end{aligned}$$

Together both equations imply $\text{proj}_{CF}(\gamma) \cdot (\gg, (t_i, \emptyset)) \in Z'_V$. Thus, Equation 5.2 needs to hold. According to Lemma 5.1, function $\text{augmentVariables}_{LN, \sigma, K}$ returns valid nodes in the search space Z_V when applied to the control-flow projection of any prefix of goal nodes. Since $\gamma \cdot (\gg, (t_i, w_i)) \in \text{prefix}(\gamma_\sigma)$, the augmentation function could have returned, e. g.:

$$\text{augmentVariables}_{LN, \sigma, K}(\text{proj}_{CF}(\gamma) \cdot (\gg, (t_i, \emptyset))) = \gamma \cdot (\gg, (t_i, w_i)).$$

This contradicts Equation 5.2 and, thus, our assumption that alignment γ_σ is not visited by Algorithm 1.

(3) *Finite number of nodes.* Suppose that Algorithm 1 does not terminate with inputs LN, M_I, M_F, σ, K . It follows that for each $q \in \mathbb{N}$, there exists a sequence of alignment moves γ_σ^q composed by q moves such that $f(\gamma_\sigma^q) \leq f(\gamma_\sigma)$. In particular, this holds for $q = \lceil \frac{f(\gamma_\sigma)}{\epsilon} + 1 \rceil$. Since each alignment move adds at least cost ϵ , $f(\gamma_\sigma^q) \geq \lceil \frac{f(\gamma_\sigma)}{\epsilon} + 1 \rceil \cdot \epsilon \geq f(\gamma_\sigma) + \epsilon$. This cannot be true since we assumed that $f(\gamma_\sigma^q) \leq f(\gamma_\sigma)$. \square

We showed that Algorithm 1 will always terminate although, depending on the minimal length of complete process traces, an arbitrary large number of nodes need to be visited. In practice, this number is kept reasonably small by the fact that models are usually designed in a way that there is no possibility to have long sequences in an alignment where each move takes on a zero cost (i. e., the corresponding arcs in the search space are associated with a cost ϵ).

Algorithm 1 Returns an Optimal Alignment

We show that the alignment returned by Algorithm 1 is, indeed, an optimal alignment as defined in Definition 4.4, i. e., any other alignment has a higher or equal cost. A direct application of A^* on the search space $Z_{LN, M_I, M_F, \sigma} = (Z_V, Z_E)$ would guarantee that the returned alignment has lower or equal cost than any other alignment in the search space, i. e., that the returned alignment is optimal. This follows from Theorem 5.1 (strictly increasing cost), the use of an admissible heuristic and the optimality of A^* [DP85]. However, we prune large parts of the search space in Algorithm 1 by only considering the nodes constructed with the optimal variable assignment instead of all possible variable assignments. Therefore, we need to show that we do not prune away a path to a goal node with lower cost.

Theorem 5.3 (Algorithm 1 returns an optimal alignment). Let $Z_{LN, M_I, M_F, \sigma} = (Z_V, Z_E)$ be an alignment search space with log trace $\sigma = \langle e_1, \dots, e_n \rangle$. Let K be a cost function. Algorithm 1 returns an optimal alignment $\gamma_\sigma \in Z_G$ with inputs LN, M_I, M_F, σ , and K :

$$\forall \bar{\gamma}_\sigma \in Z_G (g(\bar{\gamma}_\sigma) \geq g(\gamma_\sigma)). \quad \diamond$$

PROOF. Assume that Algorithm 1 would return a sub-optimal alignment $\bar{\gamma}_\sigma$. It follows that there exists another alignment $\gamma_\sigma \in Z_G (\gamma_\sigma \neq \bar{\gamma}_\sigma)$ with lower cost $g(\gamma_\sigma) < g(\bar{\gamma}_\sigma)$, which is not visited by Algorithm 1. Thus, there needs to be a node $\bar{\gamma} \in \text{prefix}(\bar{\gamma}_\sigma)$ in the search space, s.t., $g(\bar{\gamma}) > g(\gamma_\sigma)$ and node $\bar{\gamma}$ needs to be visited by Algorithm 1 before γ_σ is visited.

Algorithm 1 visits all enqueued nodes in order of increasing cost (i.e., QUEUE is a priority queue), the cost of nodes along a path in the search space is monotonically increasing (Theorem 5.1), and function $\text{augmentVariables}_{LN, \sigma, K}$ returns variable assignments that minimize the cost (Definition 5.6). Therefore, some prefix $\gamma \cdot (e, s) \in \text{prefix}(\gamma_\sigma)$ with $g(\gamma \cdot (e, s)) < g(\bar{\gamma}_\sigma)$ is not enqueued and the following needs to hold:

$$\text{proj}_{CF}(\gamma \cdot (e, s)) \notin \text{ctrlSuccessors}_{LN, M_I, \sigma}(\gamma) \quad (5.3)$$

or

$$g(\text{augmentVariables}_{LN, \sigma, K}(\text{proj}_{CF}(\gamma \cdot (e, s)))) > g(\gamma \cdot (e, s)). \quad (5.4)$$

Since $\gamma \cdot (e, s) \in \text{prefix}(\gamma_\sigma)$, we have $\text{proj}_P(\gamma \cdot (e, s)) \in TS_{N, M_I}$ and $\text{proj}_L(\gamma \cdot (e, s)) \in \text{prefix}(\sigma)$. Thus, according to Definition 5.2 it follows that $\gamma \cdot (e, s) \in Z_V$. Process traces of the original DPN LN are also process traces of the control-flow copy DPN LN' (i.e., without variables). This implies that $\text{proj}_{CF}(\gamma \cdot (e, s)) \in \text{ctrlSuccessors}_{LN, M_I, \sigma}(\gamma)$, which contradicts Equation 5.3.

Thus, Equation 5.4 needs to hold. According Lemma 5.1, the variable assignment of alignment γ_σ can be used to augment the prefix, e.g.,

$$\text{augmentVariables}_{LN, \sigma, K}(\text{proj}_{CF}(\gamma \cdot (e, s))) = \gamma \cdot (e, s).$$

An alignment with minimal cost is returned by $\text{augmentVariables}_{LN, \sigma, K}$:

$$g(\text{augmentVariables}_{LN, \sigma, K}(\text{proj}_{CF}(\gamma \cdot (e, s)))) \leq g(\gamma \cdot (e, s)),$$

which contradicts Equation 5.4 and, thus, our assumption that alignment γ_σ is not visited by Algorithm 1. \square

5.2.5 Computing an Optimal Variable Assignment

We present how the augmentation function $\text{augmentVariables}_{LN, \sigma, K}$ that is used in Algorithm 1 can be implemented as an optimization problem. Recall that in Definition 4.3, the cost function used for alignments, may assign costs to incorrect

synchronous moves, i. e., deviations in the variable assignment. The idea is to transform the augmentation with optimal variable values to the following optimization problem: Search for an variable assignment that fulfills all constraints imposed by guard expressions of the DPN and minimizes costs associated with deviations from the values recorded in the log trace.

Mixed Integer Linear Programming

We formulate this optimization problem as an Mixed Integer Linear Programming (MILP) [Sch86] problem since some of our variables are integers or can be mapped to integers (e. g., strings, booleans, timestamps) and other variables are non-integer. Other similar methods might be used to formulate this problem, e. g., constraint programming, planning, or satisfiability modulo theories. However, we choose MILP to formulate the augmentation with optimal variable assignments since it is a well-research problem and efficient open-source (e. g., LpSolve) and commercial solvers for MILP problems (e. g., Gurobi, and CPLEX) are available.

Definition 5.7 (Mixed Integer Linear Programming (MILP) Problem). Let $\text{VAR} = \{x_1, \dots, x_n\}$ be a set of variables. Let $(c_1, \dots, c_n) \in \mathbb{R}^n$ be coefficients of the objective function. Let $(b_1, \dots, b_m) \in \mathbb{R}^m$ be constants with $\forall_{1 \leq i \leq m} (b_i \geq 0)$. Let (a_{11}, \dots, a_{mn}) be constants with $\forall_{1 \leq i \leq n} \forall_{1 \leq j \leq m} (a_{ij} \in \mathbb{R})$. We define a MILP minimization ($\text{VAR}, \text{CSTR}, obj$) problem in the following form:

$$\begin{array}{llll} \text{minimize} & obj & = & c_1x_1 + \dots + c_nx_n \\ \text{subject to} & \text{CSTR} & = & a_{11}x_1 + \dots + a_{1n}x_n \leq b_1 \\ & & & \vdots \\ & & & a_{m1}x_1 + \dots + a_{mn}x_n \leq b_m \end{array}$$

with the additional constraint that for $1 \leq i \leq n$ variable x_i is either a real variable $x_i \in \mathbb{R}$ or an integer variables $x_i \in \mathbb{Z}$. The goal is to obtain values for the variables VAR such that the objective function obj is minimized. \diamond

In the remainder of this section, we need to formulate constraints of the form

$$\hat{x} = 0 \iff (a_1x_1 + \dots + a_nx_n \leq b)$$

with $\hat{x} \in \text{VAR}$ and the additional constraint $\hat{x} \in \{0, 1\}$, i. e., \hat{x} takes on value 0 if and only the constraint can be fulfilled. Such constraint can be encoded by using the following constraint [BHM77]:

$$(a_1x_1 + \dots + a_nx_n) - B\hat{x} \leq b.$$

with $B \in \mathbb{R}$ being a sufficiently large number to fulfill the constraint when $\hat{x} = 1$. The introduced variable \hat{x} serves as *indicator of the feasibility* of the constraint. Moreover,

we need to express constraints of the form $a_1x_1 + \dots + a_nx_n = b$. Such a *equality constraint* can be transformed to a set of two inequality constraints [BHM77]:

$$\begin{aligned} a_1x_1 + \dots + a_nx_n &\leq b \\ -a_1x_1 - \dots - a_nx_n &\leq -b. \end{aligned}$$

Building the Optimal Variable Assignment MILP

We show how to build a MILP problem with which we obtain the variable assignment for a sequence of alignment moves without variable assignments, i. e., a sequence γ_C as returned by function $ctrlSuccessors_{LN, M_i, \sigma}(\gamma)$ as defined in Definition 5.5. Our goal is to minimize the cost of deviations between the attribute assignments of the events and the variable assignments required by the DPN.

For the sake of a simpler presentation of the main algorithm, we assume that guards are defined in the form of MILP constraints (i. e., $a_1x_1 + \dots + a_nx_n \leq b$). Later we show how to deal with any linear guard expression. Moreover, we use helper cost function $\kappa_I : V \rightarrow \mathbb{N}$ that associates non-negative costs to incorrect variable assignments from the general alignment cost function κ (cf. Definition 4.3). In fact, for the standard cost function κ_I , we obtain $\kappa_I(v) = 1$ for all variables $v \in V$.

Input: Labeled DPN (LN), Sequence of moves (γ_C), Variable cost function (κ_I)
Result: Variables (VAR), Constraints (CSTR), Objective function (obj)

```

1 VAR ←  $\bigcup_{v \in V}(v_0)$ 
2 CSTR ←  $\bigcup_{v \in V}(v_o = in(v))$ 
3 foreach  $(e_i, s_i) \in \gamma_C$  s. t.  $s_i = (t_i, w_i) \neq \gg$  do
4   foreach  $v \in wr(t_i)$  do
5     VAR ← VAR  $\cup \{v_i, \hat{v}_i\}$ 
6     if  $e_i \neq \gg \wedge v(v) \in dom(\#(e_i))$  then
7       CSTR ← CSTR  $\cup \{\hat{v}_i \iff (v_i = \#_{v(v)}(e_i))\}$ 
8   MAP :  $V \rightarrow VAR$ 
9   foreach  $v \in V$  do
10    MAP ← MAP  $\oplus \{(v' \mapsto v_i)\}$ 
11    IDX ←  $\{i' \in \{0, \dots, i-1\} \mid v_{i'} \in VAR\}$ 
12    MAP ← MAP  $\oplus \{(v \mapsto v_{max(IDX)})\}$ 
13  CSTR ← CSTR  $\cup \{rewrite(gd(t_i), MAP)\}$ 
14 return (VAR, CSTR,  $\sum_{\hat{v}_i \in VARS} (\hat{v}_i \kappa_I(v))$ )

```

Algorithm 2: Procedure that builds a MILP to obtain an optimal variable assignment

The procedure described in Algorithm 2 is based on the method proposed in [LA13a]. Its input is a sequence γ_C without variable assignments, a DPN LN, and cost function κ_I . Its output is a MILP formulation (VAR, CSTR, obj) of the optimal variable assignment problem.

We start by creating variables and constraints for the initial value of each variable of the DPN (lines 1 and 2). For each alignment move $e_i, (t_i, w_i) \in \gamma_C$ that is a model move or (incorrect) synchronous move (line 3), we add variables and constraints for the write operations and guard expression.

For each write operation that is prescribed to happen according to function *wr* when replaying the control-flow of the process trace on the DPN two variables v_i and \hat{v}_i are added (line 5). Variable v_i captures to actual variable that is used later in the constraints. Variable \hat{v}_i serves as feasibility indicator of the observed attribute value, i. e., \hat{v}_i takes on value 1 in case the observed attribute value violates a guard of the DPN, which are transformed to MILP constraints (line 6 and 7).

Then, we add *constraints* for the guard of the transition $gd(t_i)$. We rewrite each guard with function *rewrite* s. t.:

- references to prime variable v' are replaced with the current MILP variable v_i (line 10); and
- references to variables v are replaced with the MILP variable of the previous write operation or with the initial MILP variable $v_{\max(\text{IDX})}$ (line 12).

Moreover, function *rewrite* returns \emptyset for guards that are identically true. Finally, the objective function *obj* sums the cost defined for incorrect variable assignments.

We implement function $augmentVariables_{LN,\sigma,K}$ by using the variable assignment obtained from the solution of the MILP problem. Solutions to the MILP problem minimize the cost associated with deviations between the recorded values in the log steps of γ_C and the variable assignment required by the process model. Therefore, we can use it the resulting variable assignment as a representative of an optimal variable assignment for the balanced alignment method.

In Algorithm 2, we assumed guards of the DPN to be MILP constraints as defined by Definition 5.7. However, variables of a DPN may be defined for domains different from \mathbb{Z} and \mathbb{R} . Also, guard expression may not all be of the form required by Definition 5.7. In the following two sections, we show how to transform any linear guard expression and DPN variable in a form suitable for use in Algorithm 2.

Transforming Non-linear and Non-atomic Constraints

There are two problems that need to be solved to transform a linear guard expression into MILP constraints:

1. guards containing strict inequalities ($>$, $<$) and not-equal constraints (\neq); and
2. guards containing disjunctions (\vee).

Guard expressions using these elements cannot be directly expressed as MILP constraint in the form introduce in Definition 5.7. Both problems can be solved using standard transformations [LA13a].

We encode *strict inequality constraints* of the form

$$a_{11}x_1 + \dots + a_{1n}x_n < b_1$$

by introducing a sufficiently small ϵ -value:

$$a_{11}x_1 + \dots + a_{1n}x_n - \epsilon \leq b_1.$$

For integer variables ($x_i \in \mathbb{Z}$), we use $\epsilon = 1$, for non-integer variables we need to choose a value based on the magnitude of values used in the constraint. *Not-equal constraints* $x \neq b$ can be encoded using two strict inequality constraints $x > b \wedge x < b$.

Regarding *disjunctions*, we use binary variables (i. e., integer variables constrained to $\{0, 1\}$) to formulate that at least one out of n constraints needs to hold as described in [BHM77]. For example, two constraints

$$\begin{aligned} a_{11}x_1 + \dots + a_{1n}x_n &\leq b_1 \\ \vee a_{21}x_1 + \dots + a_{2n}x_n &\leq b_2 \end{aligned}$$

are connected in disjunction by introducing binary variables y_1 and y_2 and transforming the constraints to

$$\begin{aligned} a_{11}x_1 + \dots + a_{1n}x_n - B_1y_1 &\leq b_1 \\ a_{21}x_1 + \dots + a_{2n}x_n - B_2y_2 &\leq b_2 \\ y_1 + y_2 &\leq 1. \end{aligned}$$

B_1 and B_2 need to be chosen sufficiently large such that the constraint is always fulfilled if the respective binary variable y_1 or y_2 is 1. In practice, we can choose a suitable value based on the constants used in guards and the values observed in the log.⁹

Transforming Incompatible Domains

Generally, we did not restrict the domain of DPN variables. However, in practice, we encounter variables of the following basic types, which are defined by the XES standard for event logs [IEECCIS16]:

- continuous or float (\mathbb{R}),
- integer (\mathbb{Z}),
- boolean ($\{\text{true}, \text{false}\}$),
- time ($\{1/1/1970 00:00, \dots\}$), and
- string ($\{\text{Green}, \text{White}, \text{Red}, \dots\}$).

We show how to transform values of these domains to value in the domains \mathbb{Z} or \mathbb{R} . Regarding *continuous, integer and boolean variables* the mapping is straightforward. Continuous and integer values are directly supported. Boolean values are often transparently mapped to the domain $\{0, 1\}$ by the employed MILP library. We describe the mapping for time values and string literals in the following two paragraphs.

⁹Please note that we cannot simply choose B_i to be infinite since this would lead to numerical instabilities due to the use of fixed-precision arithmetic in most MILP solvers.

String literals. We build a mapping table between string literals and positive integers (i. e., \mathbb{N}_0). The mapping table can be obtained since the number of string literals in the event log and in guard expressions is finite. We implement all comparison operations using such an integer representation of string literals.

Time values. Converting time values to integer numbers using the UNIX timestamp encoding, i. e., the number of milliseconds since 1970, is straightforward. However, this conversion may result in computations involving numbers of several magnitudes of difference, e. g., $1483239500 \leq 1481539500 + 3600000$ would be the computation for a UNIX timestamp encoding of the guard: $\text{time}'_{\text{che}} \leq (\text{time}_{\text{che}} + 1\text{hr})$. Constants of several magnitudes difference cause numerical instabilities in MILP solvers [Mar03], which lead to wrong solutions being returned. We solve this problem by expressing time constraints relative to the start-time of the process instance and by using a suitable time unit. For example, in guard $\text{time}'_{\text{che}} \leq (\text{time}_{\text{che}} + 1\text{hr})$, we express the relative time in hours instead of milliseconds.

Example 5.2 (Building the MILP problem for the optimal variable assignment). Assume that we want to find the optimal variable assignment for the sequence of alignment moves γ_8 , which was introduced in Figure 5.5. Before augmentation with variable assignments, we denote the sequence as γ_C^8 . In fact, the alignment moves γ_C^8 are generated by Algorithm 1 as part of its search space, which is shown in Figure 5.5. Please note that all variable assignments are missing since γ_C^8 still needs to be augmented with an optimal variable assignment:

$$\gamma_C^8 = \langle (e_{30}, (t_{\text{tri}}, \emptyset)), (e_{31}, (t_{\text{reg}}, \emptyset)), (\gg, (t_{\tau_1}, \emptyset)) \rangle.$$

We apply Algorithm 2 using γ_C^8 as input. Algorithm 2 builds the following MILP problem:

$$\text{minimize} \quad \widehat{\text{data}}_{\text{color}}^0 + \widehat{\text{time}}_{\text{reg}}^0 \quad (5.5)$$

$$\text{subject to} \quad \text{data}_{\text{color}}^0 - B_1 \widehat{\text{data}}_{\text{color}}^0 \leq 1 \quad (5.6)$$

$$-\text{data}_{\text{color}}^0 - B_1 \widehat{\text{data}}_{\text{color}}^0 \leq -1 \quad (5.7)$$

$$\text{time}_{\text{reg}}^0 - B_2 \widehat{\text{time}}_{\text{reg}}^0 \leq 15 \quad (5.8)$$

$$\text{time}_{\text{reg}}^0 - B_2 \widehat{\text{time}}_{\text{reg}}^0 \leq -15 \quad (5.9)$$

$$\text{data}_{\text{color}}^0 - \epsilon - B_3 \text{or}_1 \leq 1 \quad (5.10)$$

$$-\text{data}_{\text{color}}^0 + \epsilon - B_3 \text{or}_2 \leq -1 \quad (5.11)$$

$$\text{or}_1 + \text{or}_2 \leq 1. \quad (5.12)$$

The first alignment move $(e_{30}, (t_{\text{tri}}, \emptyset))$ is a synchronous move. Therefore, we need to check for possible variable assignments (Algorithm 2, line 3). Transition

t_{tri} writes one variable $data_{color}$, i.e., $wr(t_{tri}) = \{data_{color}\}$, for which we add two MILP variables $data_{color}^1$ and \widehat{data}_{color}^1 to the set of MILP variables (Algorithm 2, line 5). Then, we record the value of mapped attribute $color$ (i.e., $v(data_{color}) = color$) that was observed in e_{30} (Algorithm 2, line 6-7). Since e_{30} recorded the color *White*, which we assume to be mapped to the integer 1, we add the constraints 5.6 and 5.7. Moreover, the decision variable \widehat{data}_{color}^0 is added to the objective function in 5.5. The coefficient used in the objective function is $\kappa_1(v) = 1$. Since the guard expression associated with t_{tri} is $gd(t_{tri}) = \text{true}$ there are no constraints that need to be added.

We proceed with the next alignment move $(e_{31}, (t_{reg}, \emptyset))$. Again, one variable is written: $time_{reg}$; thus, the constraints 5.8 and 5.9 are added. The value of the time attribute of e_{31} is transformed to the integer 15 because event e_{31} occurred 15 minutes after start of the trace.

Finally, with the last alignment move $(\gg, (t_{\tau_1}, \emptyset))$, two constraints are added for a guard expression. The guard $data_{color} \neq \text{White}$ is transformed to the constraints 5.10, 5.11, and 5.12. A possible solution to the obtained MILP problem is variable assignment:

$$\begin{aligned}\widehat{data}_{color}^0 &= 0 & \widehat{time}_{reg}^0 &= 0 \\ data_{color}^0 &= 1 & time_{reg}^0 &= 15.\end{aligned}$$

The observed value 2 (*White*) for attribute $color$ does not satisfy the guard expression $data_{color} \neq \text{White}$. Therefore, the MILP solver suggests a different value: 0 (*Green*). There are multiple optimal solutions, e.g., value 3 (*Red*) would also minimize the objective function. However, one of the optimal solutions will be picked and, thus, the search space is pruned considerably. The resulting alignment moves are:

$$\begin{aligned}\gamma_8 = & ((e_{30}, (t_{tri}, (data_{color} \mapsto \text{Green}))), \\ & (e_{31}, (t_{reg}, (time_{reg} \mapsto 12/12/16 10:45))), \\ & (\gg, (t_{\tau_1}, \emptyset))).\end{aligned}$$

In summary, the overall cost is $K(\gamma^8) = 1$ due to the incorrect attribute value recorded by event e_{30} .

5.2.6 Computational Complexity

We now discuss the computational complexity of the balanced alignment method from two angles. First, we show that the worst-case complexity of our method is double exponential. Second, we show that restricting the input to relaxed data sound DPN is necessary since the problem of deciding whether an alignment exists is undecidable for arbitrary DPNs.

Complexity of Alignments for Relaxed Data Sound DPN

The worst-case complexity of our A^{*}-based algorithm is exponential in the length of the path that leads from the initial search-space node to the nearest goal node [DP85]. Applied to the problem of finding an optimal alignment between a relaxed sound DPN and an log trace, this means that the worst-case complexity is exponential in the length of the alignment. This is of the same order of magnitude as the log-trace traces, under two assumptions: (1) on average, each trace event is associated with one or two moves (e. g., a move in both or a move in log plus a move in model) and (2) the number of invisible routing transitions in process traces of the DPN that are required to reach the final marking is negligible in comparison to the number of transitions representing visible activities.

For each node that is visited, a MILP problem needs to be solved. So, the number of problems to be solved is exponential in the length of log trace. The worst-case complexity of solving an MILP problem is exponential in the number of variables and constraints, which, in our setting, is translated to the number of variables written by and guards associated to transitions. We conclude that the worst-case complexity is double exponential in the length of the trace.

Decidability of the General Case

So far, we have required relaxed data sound DPN as input for the balanced alignment method. We show that this restriction is necessary since DPN is a turing complete notation and, thus, the alignment problem for arbitrary DPN is undecidable. This result highlights that DPNs are a powerful notation and that there are limitations to our balanced alignment technique for arbitrarily complex DPN specifications.

We use an extended class of Petri nets, so called *Inhibitor nets* [Mur89], in which an additional type of arc is introduced: inhibitor arcs. An inhibitor arc connects a place to a transition and changes the execution semantics as follows. A transition connected to an inhibitor arc can only fire if the input place with the inhibitor arc has no token and all other input places have at least one token [Mur89]. Inhibitor nets are known to be turing complete [Bus02; Min67].

Theorem 5.4 (DPNs are Turing complete). There exists a DPN $N = (P, T, F, V_P, \text{dom}, \text{in}, \text{wr}, \text{gd})$ with initial marking M_I and final marking M_F such that N simulates any turing machine, i. e., DPN are *turing complete*. \diamond

PROOF. Figure 5.6 shows how to simulate an inhibitor arc using one variable and guards in a DPN. For each place p that is connected with an inhibitor arc to a transition t , we introduce a counter variable X_p that tracks the number of tokens currently stored in p . Then, we can simulate the zero testing semantics of the inhibitor arc by assigned the guard $X_p = 0$ to transition t . The behavior modeled

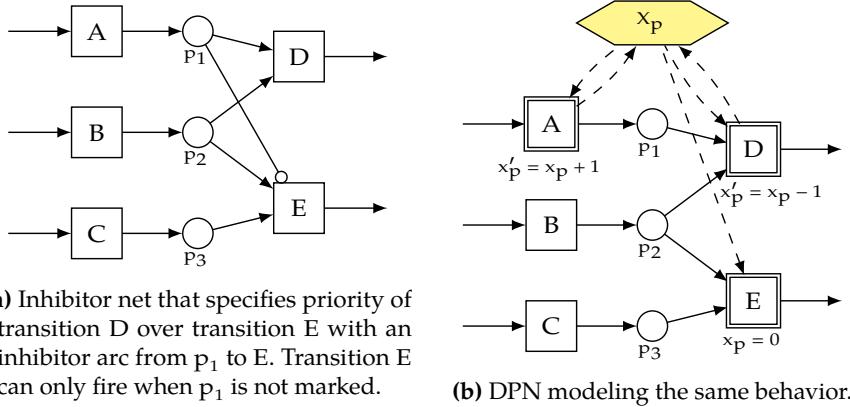


Figure 5.6: Simulation of inhibitor arcs using variables and guards of a DPN

in Figure 5.6 requires the use of inhibitor arcs, i. e., it cannot be expressed with a standard Petri net. DPNs are at least as expressive as inhibitor nets. Thus, DPNs are turing complete. \square

Theorem 5.5 (The alignment problem for arbitrary DPNs is undecidable). Let $L = (E, \Sigma, \#, \mathcal{E})$ be an event log. Let (N, λ, v) be any labeled DPN with initial marking M_I and final marking M_F . Let $LTS = (TS_{N, M_I, M_F}, \lambda, v)$ be the labeled trace set induced by the labeled DPN. The *alignment problem* for DPNs is the following decision problem: Given any log trace $\sigma \in \mathcal{E}$, decide whether there is an alignment between σ and the DPN, i. e., $\exists \gamma_\sigma \in \gamma_{L, LTS}^*$. The *alignment problem is undecidable*. \diamond

PROOF. Let L be an event log containing the empty trace $\sigma_\emptyset = \langle \rangle$. Assume that we could decide for the empty trace if there is an alignment $\gamma_{\sigma_\emptyset} \in \gamma_{L, LTS}^*$. The projection of alignment $\gamma_{\sigma_\emptyset}$ on the process steps is required to be a process trace of the trace set LTS. Trace set LTS contains all process traces of the DPN that start in the initial state and end in one of the final states. Thus, by computing an alignment, we could decide whether there is, indeed, a firing sequence in the DPN that starts in the initial state with marking M_I and ends in one of the final states with marking M_F . Deciding whether there is such a firing sequence is equivalent to deciding whether the DPN token-game halts. According to Theorem 5.4, DPNs are turing complete. Therefore, we could decide the halting problem for turing machines. However, the halting problem for turing machines is undecidable [Tur37], thus, leading to a contradiction. \square

Whereas the general problem is undecidable for arbitrary DPNs, we would like to highlight that for practical applications our alignment method is often feasible. It uses the finite log traces to guide the search for an alignment. Therefore, we assume

that our method is applied to process models that terminate after a finite number of steps and that the length of typical process traces is close to the length of the log traces.

5.2.7 Optimizations

We have shown how to compute balanced alignments between multi-perspective models and traces of an event log. As discussed, computing a fully balanced alignment is undecidable for arbitrary DPNs and a computationally difficult problem for relaxed data-sound DPN. Therefore, we cannot avoid an exponential worst-case complexity. Nevertheless, efficiency for the average case that is encountered in practice is of the utmost importance. We describe two optimizations, which speed up the computation and limit the number of nodes to be visited:

- we elaborate on the heuristic function used to guide the A* search and
- we prune the search space using an equivalence relation for nodes in Z_V .

Guiding the Search With a Heuristic Function

We use the same heuristic function h that was introduced in [Adr14] for control-flow only alignments. This heuristic exploits the Petri-net marking equation to rule out most nodes for which all goal states have become unreachable.¹⁰ A formal introduction of the marking equation and this heuristic is out of scope for this thesis. We limit ourselves to argue that a heuristic developed for control-flow alignments is also admissible when adding other perspectives.

First, we show that the cost of an alignment between a log trace and a DPN with guard expressions and variable assignments is always higher or equal to the cost of an alignment between the same log trace and a copy of the DPN without guard expressions and variable assignments.

Lemma 5.2 (Variable, guards, and write operations increase the alignment cost). Let $Z_{LN, M_I, M_F, \sigma} = (Z_V, Z_E)$ be an alignment search space between log trace σ and LN. Let $Z'_{LN', M_I, M_F, \sigma} = (Z'_V, Z'_E)$ be the alignment search space based on the control-flow copy DPN LN' . Let K be a cost function. The variables, guards, and write operations added in LN may only increase the alignment cost compared to LN' :

$$\forall \gamma \in Z_V \forall \bar{\gamma} \in Z'_V (proj_{CF}(\gamma) = proj_{CF}(\bar{\gamma})) \implies (K(\gamma) \geq K(\bar{\gamma})). \quad \diamond$$

PROOF. Cost function $K(\gamma) \in \mathbb{R}$ as defined in Definition 4.3 is required to assign a non-negative cost to any deviation. The only difference between the alignment moves in γ and the alignment moves in $\bar{\gamma}$ are the variable assignments. Thus, it

¹⁰The marking equation provides a necessary but not sufficient condition for the reachability of the goal state.

holds that $K(\text{proj}_{\text{CF}}(\gamma)) = K(\text{proj}_{\text{CF}}(\bar{\gamma}))$, i.e., both alignments have the same cost for deviations based on the control-flow only. Cost function K *only adds additional costs* for deviations with respect to other perspectives. Such deviations can only be caused by incorrect variable assignment operations, for which we require a *non-negative cost*. Thus, $K(\gamma) \geq K(\bar{\gamma})$. \square

A heuristic that ignores variable assignments and guards of the DPN might provide a suboptimal underestimate. Metaphorically speaking, some roads predicted to have a low remaining path cost might be blocked by guards, which are not taken into account. However, any path that is predicted to have high costs regarding the control-flow will have at least the same cost when regarding all perspectives, i.e., no shortcuts can be built using variable assignments and guards.

Proposition 5.1 (The heuristic function introduced in [Adr14] is admissible). Let $Z_{\text{LN}, M_I, M_F, \sigma} = (Z_V, Z_E)$ be an alignment search space between log trace σ and LN. Let K be a cost function. Let $g(\gamma) = K(\gamma) + \epsilon |\gamma|$ be the cost function. Let h be the heuristic function of [Adr14]. Function h is admissible for the alignment search space, i.e., it underestimates the required cost to reach any goal node γ_σ from any node γ :

$$\forall_{\gamma \in Z_V} \forall_{\gamma_\sigma \in Z_G} (\gamma \in \text{prefix}(\gamma_\sigma)) \implies (h(\gamma) \leq g(\gamma_\sigma) - g(\gamma)). \quad \diamond$$

PROOF. From [Adr14] we know that the heuristic function h is admissible for control-flow alignments, i.e., $\forall_{\gamma \in Z_B} \forall_{\gamma_\sigma \in Z_G}$ if $\gamma \in \text{prefix}(\gamma_\sigma)$, then, it follows that

$$\begin{aligned} h(\gamma) &\leq g(\text{proj}_{\text{CF}}(\gamma_\sigma)) - g(\text{proj}_{\text{CF}}(\gamma)) \\ \Leftrightarrow h(\gamma) &\leq K(\text{proj}_{\text{CF}}(\gamma_\sigma)) - K(\text{proj}_{\text{CF}}(\gamma)) + \epsilon \cdot (|\gamma_\sigma| - |\gamma|). \end{aligned} \quad (5.13)$$

Assume that the heuristic function would not be admissible for a multi-perspective alignment, i.e.:

$$\begin{aligned} g(\gamma_\sigma) - g(\gamma) &< h(\gamma) \\ \Leftrightarrow K(\gamma_\sigma) - K(\gamma) + \epsilon \cdot (|\gamma_\sigma| - |\gamma|) &< h(\gamma). \end{aligned} \quad (5.14)$$

Let $\gamma_\sigma = \gamma \cdot \bar{\gamma}$. Since cost function K is additive it follows that:

$$\begin{aligned} K(\gamma_\sigma) &= K(\gamma) + K(\bar{\gamma}) \\ \Leftrightarrow K(\bar{\gamma}) &= K(\gamma_\sigma) - K(\gamma) \\ &\text{and} \\ K(\text{proj}_{\text{CF}}(\gamma_\sigma)) &= K(\text{proj}_{\text{CF}}(\gamma)) + K(\text{proj}_{\text{CF}}(\bar{\gamma})) \\ \Leftrightarrow K(\text{proj}_{\text{CF}}(\bar{\gamma})) &= K(\text{proj}_{\text{CF}}(\gamma_\sigma)) - K(\text{proj}_{\text{CF}}(\gamma)). \end{aligned}$$

Therefore, the heuristic function needs to overestimate the cost of $\bar{\gamma}$ (Equation 5.14):

$$K(\bar{\gamma}) + \epsilon \cdot |\bar{\gamma}| < h(\gamma)$$

However, we know that the heuristic function underestimates the cost for a control-flow alignment (Equation 5.13):

$$K(\bar{\gamma}) + \epsilon \cdot |\bar{\gamma}| < h(\gamma) \leq K(proj_{CF}(\gamma_\sigma)) - K(proj_{CF}(\gamma)) + \epsilon \cdot (|\gamma_\sigma| - |\gamma|)$$

Since the cost function is additive, we have:

$$\begin{aligned} K(\bar{\gamma}) + \epsilon \cdot |\bar{\gamma}| &< K(proj_{CF}(\bar{\gamma})) + \epsilon \cdot (|\bar{\gamma}|) \\ \Leftrightarrow K(\bar{\gamma}) &< K(proj_{CF}(\bar{\gamma})) \end{aligned}$$

Thus, our assumption in Equation 5.14 contradicts Lemma 5.2, i.e., our cost function K only adds additional costs for deviations with respect to other perspectives. \square

Pruning Equivalent States in Z_V

Algorithm 1 explores the graph $Z_{LN, M_I, M_F, \sigma} = (Z_V, Z_E)$ by conducting an A* search to find an optimal alignment. However, the search graph Z explored by Algorithm 1 is, in fact, a tree. Since search nodes in Z_V are sequences of alignment moves and edges in Z_E only add moves, the search never encounters a node that was already visited. When applied on a search graph, the A* algorithm can make use of two optimizations to reduce the number of explored nodes:

1. maintaining a closed set of visited nodes to prevent nodes equivalent to already visited nodes to be explored; and
2. checking whether an equivalent node with lower cost is already enqueued in the queue before adding a new node.

Algorithm 3 is an optimized variant of Algorithm 1 that employs these two optimizations. In comparison to Algorithm 1, we added:

- the closed set $CLOSED$ and use function $isClosed : Z'_V \rightarrow \{\text{true}, \text{false}\}$ to check whether an equivalent node has already been visited, and
- the function $isBetter : (Z_V \times \text{QUEUE}) \rightarrow \{\text{true}, \text{false}\}$ to keep only nodes with the lowest cost among all equivalent nodes in the queue.

We can use Algorithm 3 instead of Algorithm 1 to prune away large portions of the search space. We define an equivalence relation:

$$isSame \subseteq Z'_V \times Z'_V$$

on states in Z'_V and implement functions $isClosed$ and $isBetter$ as:

$$isClosed(CLOSED, \gamma) = \begin{cases} \text{true} & \text{if } \exists_{\bar{\gamma} \in CLOSED} ((\gamma, proj_{CF}(\bar{\gamma})) \in isSame) \\ \text{false} & \text{otherwise,} \end{cases}$$

```

Input: Labeled DPN (LN), Initial and final markings ( $M_I, M_F$ ), Log trace ( $\sigma$ ), Cost function ( $K$ )
Result: Balanced alignment ( $\gamma$ )
1  $\gamma \leftarrow \langle \rangle$ 
2 QUEUE =  $\langle \rangle$ 
3 CLOSED =  $\emptyset$ 
4 while  $\neg(\gamma \text{ is an alignment})$  do
5   CLOSED  $\leftarrow$  CLOSED  $\cup \{\gamma\}$ 
6   foreach  $\gamma_C$  in  $ctrlSuccessors_{LN, M_I, \sigma}(\gamma)$  s. t.  $\neg isClosed(CLOSED, \gamma_C)$  do
7      $\gamma_D \leftarrow augmentVariables_{LN, \sigma, K}(\gamma_C)$ 
8     if  $\gamma_D \neq \top$  then
9        $g(\gamma_D) \leftarrow K(\gamma_D) + \epsilon |\gamma_D|$ 
10       $f(\gamma_D) \leftarrow g(\gamma_D) + h(\gamma_D)$ 
11      if  $isBetter(QUEUE, \gamma_D)$  then
12        enqueue(QUEUE,  $\gamma_D, f(\gamma_D))$ 
13    $\gamma \leftarrow pollLowestCost(QUEUE)$ 
14 return  $\gamma$ 

```

Algorithm 3: Optimized procedure that computes a balanced alignment

and

$$isBetter(QUEUE, \gamma) = \begin{cases} \text{false} & \text{if } \exists_{\bar{\gamma} \in \text{QUEUE}} (((proj_{CF}(\gamma), proj_{CF}(\bar{\gamma}) \in isSame \\ & \quad \wedge f(\bar{\gamma}) \leq f(\gamma))) \\ \text{true} & \text{otherwise.} \end{cases}$$

In both cases only one of equivalent states in the search space needs to be considered to obtain an optimal solution. The search space is pruned.

We implement *isSame* as follows. Any two nodes in the search space are equivalent according to *isSame* if and only if:

1. the *same sequence of events* is recorded in their log projections;
2. the *same marking* is reached in their process projections;
3. the *same sequence of guard expressions* needs to be fulfilled in their process projections; and
4. the *same sequence of alignment moves that include a process step with write operations* is recorded.

This implementation of *isSame* leads to a considerable reduction of the search space that needs to be explored. Often, process models only include guard expressions and write operations on a few of the transitions. For these models, re-orderings of moves (e. g., in parallel branches) that are not associated with guards or write operations do not need to be explored.

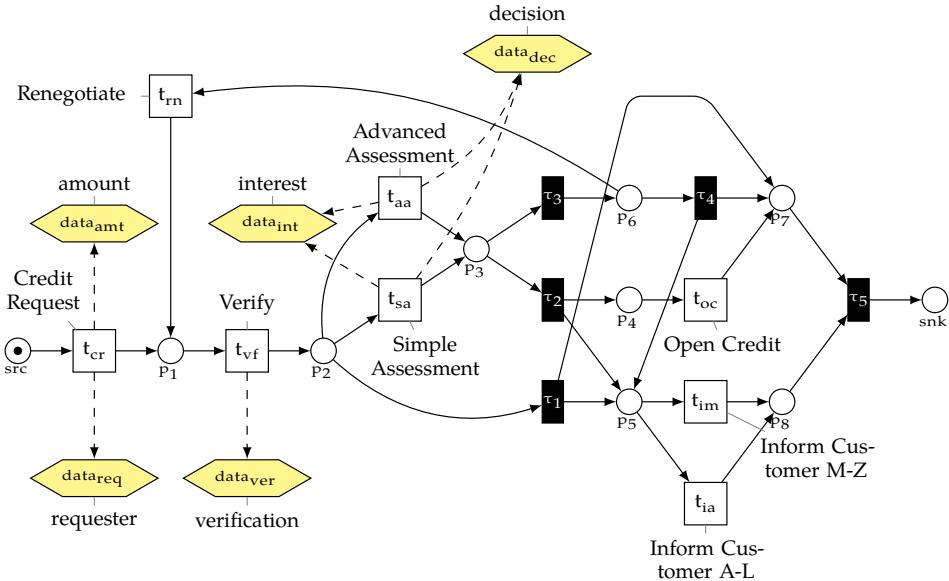
We argue briefly that our implementation of *isSame* does not influence the optimality of the result returned by Algorithm 3. The *sequence of events* in the log projection uniquely defines the remaining events that need to be aligned. The *marking uniquely defines the possible continuations* in the Petri net underlying the DPN and, thus, the set of possible firing sequences to reach a final marking when considering control-flow only. Conditions (1) and (2) were already used in [Adr14] to define equivalent states for alignments between the control-flow of a process model and an event log. However, our search space is defined for multi-perspective alignments. Variable assignments and guard expressions influence the successor nodes of a node in the search space $Z_{LN, M_1, M_F, \sigma}$ of the original DPN LN. Therefore, we also need to consider the set of possible variable assignment for successor nodes in our implementatin of *isSame*.

In Section 5.2.5, we introduced a MILP problem formulation to obtain the optimal variable assignment. Intuitively, if the same MILP problem needs to be solved for both nodes, the set of possible variable assignments is the same for both nodes. As described in Algorithm 2 the MILP problem formulation depends on (a) the sequence of write operations and (b) the sequence of process transitions that are associated with guard expressions, which need to be fulfilled. Therefore, we add conditions (3) and (4) which compare sequence of transitions with associated guard expressions and the sequence of alignment moves with write operations for both nodes. Condition (3) ensures that the constraints due to guard expressions are equal for both nodes. Condition (4) ensures that the same values have been recorded for variables. Together both conditions guarantee that the same MILP problem formulation is obtained for nodes deemed equivalent by relation *isSame*. Thus, we can use relation *isSame* to prune the search space as described in Algorithm 3.

5.3 Evaluation

We used both real-life and synthetic data sets to evaluate the usefulness and feasibility of the balanced approach. In the interest of a clear separation between the more theoretical contributions of this thesis and their applications to case studies, we describe the evaluation on real-life data sets separately as part of the presentation of case studies in Sections 12.2, 13.2 and 15.3.

In the following sections, we focus on the evaluation with controlled, synthetic examples. We assessed the *effectiveness* and the *efficiency* of the method. Regarding the effectiveness, we compared results returned by the **balanced approach** with those returned by the **non-balanced** one from [LA13a]. Regarding the efficiency, we show that our implementation can compute the alignment of deviating traces of considerable length and with large number of deviations in a reasonable amount of time given the complexity of the problem. Clearly, we cannot expect an efficient worst-case computation time since the worst case complexity of our method is exponential.



(a) The labeled DPN structure of the credit card application process.

Transition	Guard expression
τ_1	$\text{data}_{\text{ver}} = \text{false}$
τ_2	$\text{data}_{\text{dec}} = \text{true}$
τ_3	$\text{data}_{\text{dec}} = \text{false}$
t_{aa}	$\text{data}_{\text{ver}} = \text{true} \wedge \text{data}_{\text{amt}} > 5000 \wedge 0.1 < (\text{data}'_{\text{int}} ?? \text{data}_{\text{amt}}) < 0.15$
t_{sa}	$\text{data}_{\text{ver}} = \text{true} \wedge \text{data}_{\text{amt}} \leq 5000 \wedge 0.15 < (\text{data}'_{\text{int}} ?? \text{data}_{\text{amt}}) < 0.2$
t_{oc}	$\text{data}_{\text{dec}} = \text{true} \wedge \text{data}_{\text{ver}} = \text{true}$
t_{im}	$\text{data}_{\text{req}} \geq "M"$
t_{ia}	$\text{data}_{\text{req}} \leq "L"$
t_{rn}	$\text{data}'_{\text{amt}} \leq \text{data}_{\text{amt}}$

(b) Guards assigned to transitions that are not always enabled (i.e., $gd(t) \neq \text{true}$).

Figure 5.7: Example credit card application process used to evaluate the balanced alignment. The model is based on the model presented in [LA13a; Man+16a] and contains a loop, parallelism, as well as complex guard expressions.

5.3.1 Datasets and Experimental Setup

As synthetic datasets, we used two simulated process models:

1. the hospital model that was introduced in Figure 3.4 and
2. a synthetic model of a credit card application process based on previous work [LA13a; Man+16a] as shown in Figure 5.7.

Both models contain several guards and write operations regarding the data, time, and resource perspective. Moreover, they contain loops, parallelism, and choices in the control-flow perspective. Therefore, both models are suited as representatives of typical multi-perspective process models that our method is applicable to.

We generated synthetic event logs using CPN Tools [JKW07]. The resulting logs allow for various controlled experiments to evaluate the performance of the approach. In total, we generated three event logs of 5,000 traces for each of the two processes. The event logs contain traces of varying length, i. e., between 3 and 95 events. Then, we created several test logs with introduced artificial noise. We test different types of noise by creating test logs for each of the following operations:

- invalidating data assignments with regard to the guards,
- swapping events,
- adding artificial events, and
- removing events.

For both processes and for each type of noise, we create test logs with increasing levels of noise, i. e., 2%, 5%, and 10% of noise. Introducing $x\%$ of noise in a perfectly fitting log means that we manipulate the event logs by applying the respective type of noise to $x\%$ of the events in each trace. We always invalidate $x\%$ of the data assignments and obtain 9 events logs with 2%, 5%, and 10% of swapped, added, and removed events respectively for both of the processes.

We used 18 test logs to test the efficiency and effectiveness of our alignment method. We aligned each of the 18 generated logs to the respective process model. For all synthetic experiments we employed the standard cost function K_1 . We repeated each performance experiment three times and took the minimum computation time to eradicate external factors such as garbage collection from distorting the experiment. All experiments were executed on a standard quad-core laptop with 16 GB of memory. The reported computation times correspond to the execution of our method on a single computation thread. Then, we validated the following three statements about our method:

- the computation time of the balanced alignment per trace grows exponential in the length of traces and the exponential factor is influenced by the level and type of deviations (Figures 5.8 and 5.9),
- the optimizations to the balanced alignment method in Section 5.2.7 improve the performance considerably (up to several magnitudes, see Figure 5.10), and

- the balanced alignment provides better alignments (i. e. optimal ones) compared to the non-balanced approach described in [LA13a] (Figure 5.11).

5.3.2 Results

We investigated the performance (efficiency) and the effectiveness of the method.

Efficiency

Figures 5.8 and 5.9 illustrate the execution time for alignments between the credit application and the hospital process example. There are 9 scattered plots: one plot for each combination of noise level and noise type. Dots represent the median of the computation time in milliseconds required for the alignment of traces of length x . The computation times may differ between traces of the same length since the introduced noise might affect them differently. In some cases more deviations lead to a more complex alignment problem for the same trace length.

Three series of dots are displayed: green dots refer to the execution time of the naïve balanced alignment only using the control-flow heuristic for A^* ; orange dots refer to the fully optimized balanced alignment; and purple dots refer to the non-balanced alignment approach described in [LA13a]. We also tried to run the experiment without the A^* heuristic (i. e., $h=0$ for all nodes), however, the computation did not finish within a day for one of the combinations of noise type and noise level. We limited the alignment computation to 30 minutes (dotted line at 1,800,000 ms) per trace to limit the overall time needed for the experiments. Moreover, we considered a computation time of more than 30 minutes per trace to be unreasonable for practical applications. Therefore, for some traces, the reported computation time represents a lower bound of the real computation time. Except for a small number of cases (11 traces) the optimized method was able to compute the alignment within 30 minutes, whereas the naïve method fails to compute alignment within 30 minutes for most longer traces (141 traces).

Regarding the different types of noise, it appears that removing tasks and swapping tasks results in more difficult alignment problems. This can be explained for both models by looking at the guard expressions. In fact, the guard of t_{rn} (Renegotiate) links the old value of $data_{amt}$ to its new value: $data'_{amt} \leq data_{amt}$ and the guards of transitions t_{aa} (Advanced Assessment) and t_{sa} (Simple Assessment) connect the value of the variable $data_{amt}$ with the value of the variable $data_{int}$ (cf., Section 5.3). Moreover, in the hospital process the guard $time'_{check} \leq time_{check} + 1h$ connects the values for variable $time_{check}$ of events that occurred early in the trace with those that occurred late in the trace. Thus, when swapping or removing events the written values do not match the expected values and the balanced alignment often needs to backtrack to consider all possible explanations for the observed invalid values.

We compared the results of the balanced and the non-balanced alignment, as well as the results of the naïve and the optimized method. For all methods, the

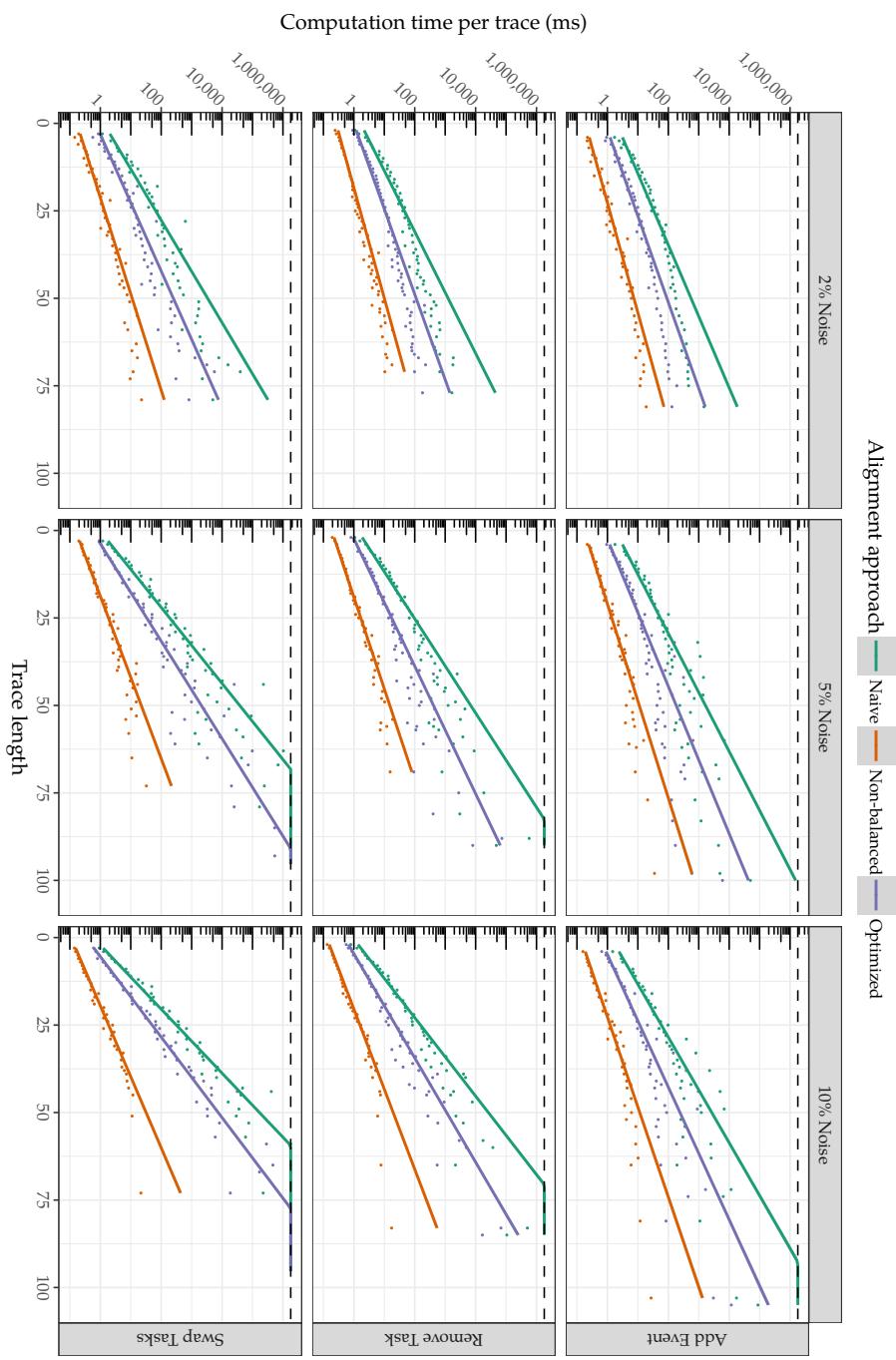


Figure 5.8: Computation time of the naïve, optimized, and non-balanced alignment of the credit process, using varying trace length and noise types. The staged approach is unable to compute a solution for some traces.

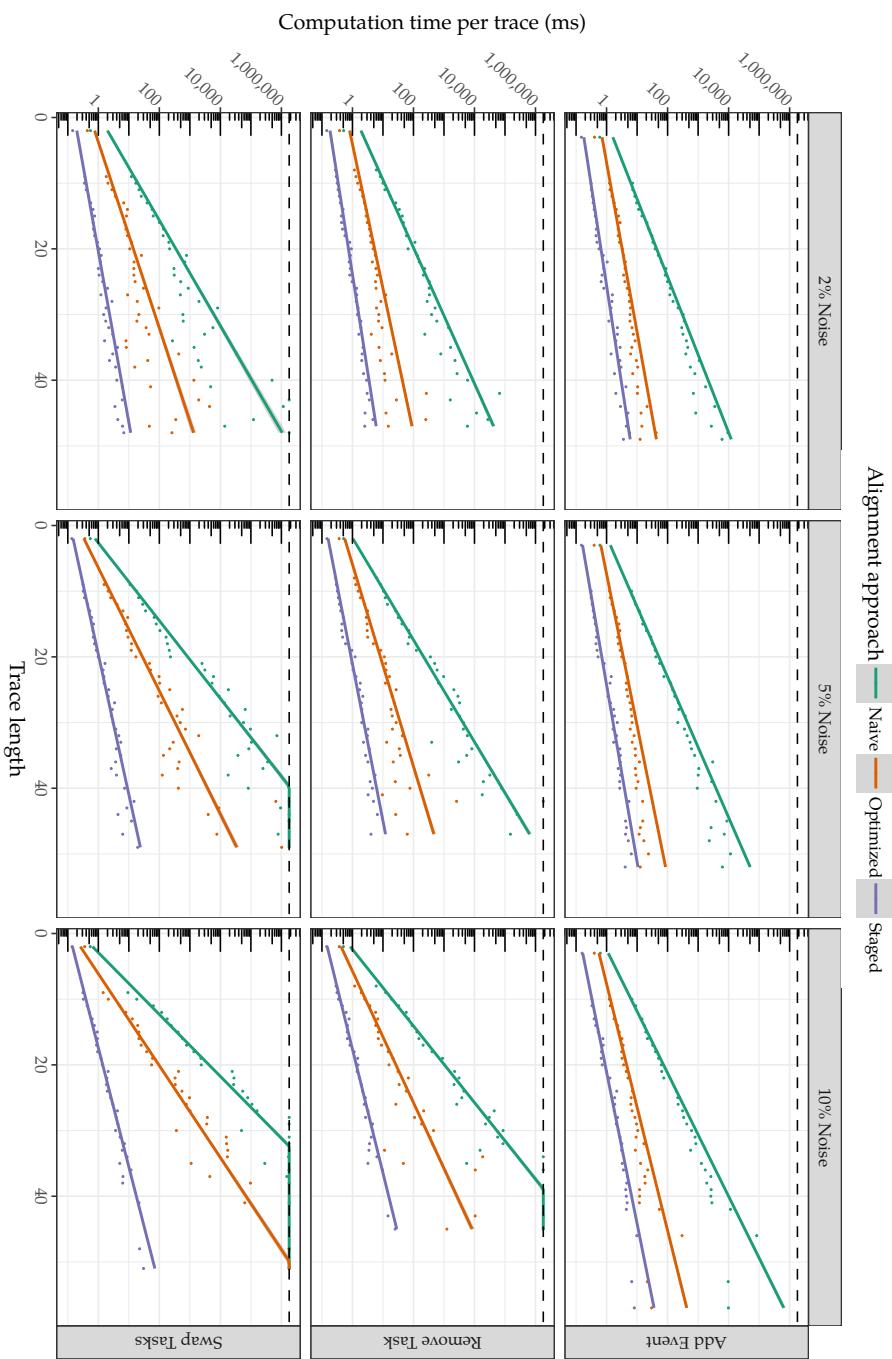


Figure 5.9: Computation time of the naïve, optimized, and non-balanced alignment of the hospital process, using varying trace length and noise types.

execution time grows exponentially in the length of the input traces. However, the balanced method is computationally more expensive than the non-balanced method. Whereas the difference for short traces is in the order of one magnitude, it increases to several orders of magnitudes for longer traces. Moreover, the non-balanced method fails to compute alignments for some longer traces with 5% and 10% of noise, which highlights the need for the balanced method.

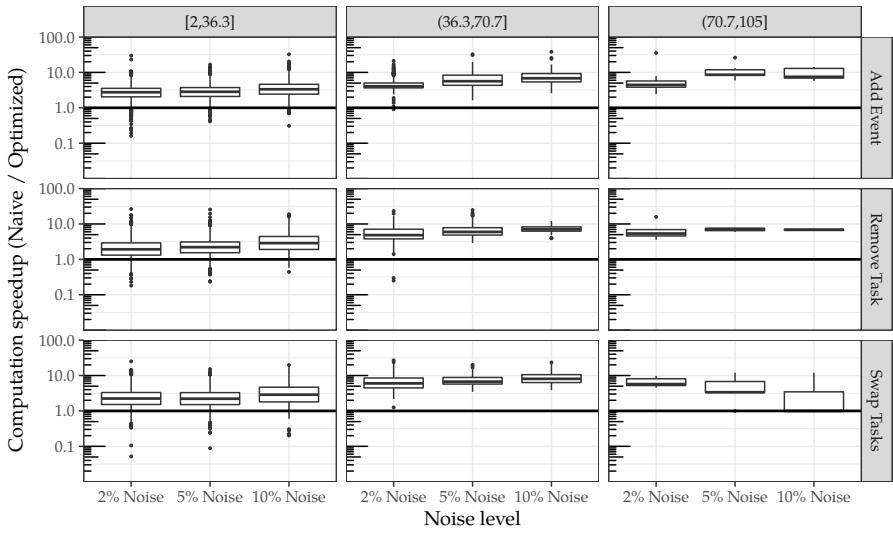
Balanced vs. non-balanced method. The non-balanced alignment can be obtained in less than 100 milliseconds for most of the traces. Unlike the balanced method presented here, the non-balanced alignment method proposed in [LA13a] separates the control-flow alignment from the alignment of the other perspectives (i.e., it follows a staged approach). Therefore, the non-balanced alignment is several magnitudes faster than the balanced alignment presented in this thesis. However, the non-balanced alignment method does not guarantee to return an optimal solution wrt. a cost function defined for all perspectives.

Naïve vs. optimized method. For short traces and for a low level of noise the naïve balanced alignment method is feasible. However, by employing the optimizations we are able to compute balanced alignments for long traces and higher levels of noise about a magnitude faster than with the naïve method for the credit process and about two magnitudes faster for the hospital process. Thus, the optimizations enable alignments of long traces with computation times feasible in real-life settings.

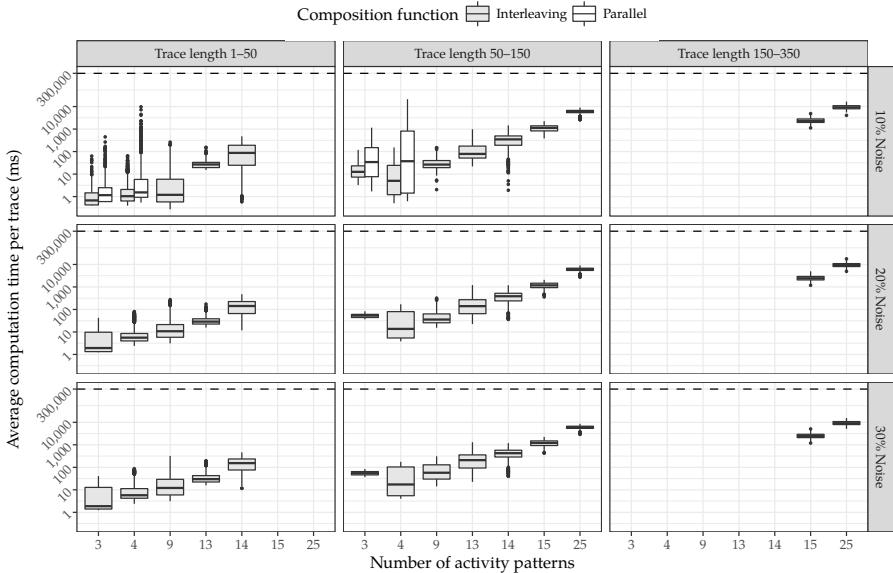
Figures 5.10a and 5.10b give a more detailed overview of the computation time speedup that the optimization in Section 5.2.7 provides for the different combinations of noise level and noise type. For longer log traces, the optimized variant is around one magnitude times faster than the naïve method in the credit process and around two magnitudes faster for the hospital process. In some cases the speed-up is even up to three magnitudes. It seems that the optimizations do not lead to a speed-up for 5% and 10% of noise and the noise type *swap task*. However, this is because we limited the execution time per trace to 30 minutes. In fact, for a lot of traces the naïve method could not determine an alignment within this time limit. In some cases the optimized method took longer than the naïve method. We assume that this is due to random garbage collection processes or interference by the operating system that were not removed by repeating the experiment three times. For very short cases it may be due to the overhead imposed by checking the state equivalence as defined in Section 5.2.7.

Effectiveness

An obvious next question is: *Are better alignments justified by the increase of computation time?* We excluded the traces whose fitness is 1.0 from this discussion, since



(a) Credit process



(b) Hospital process

Figure 5.10: Computation speedup for different types of noise and varying levels of noise.

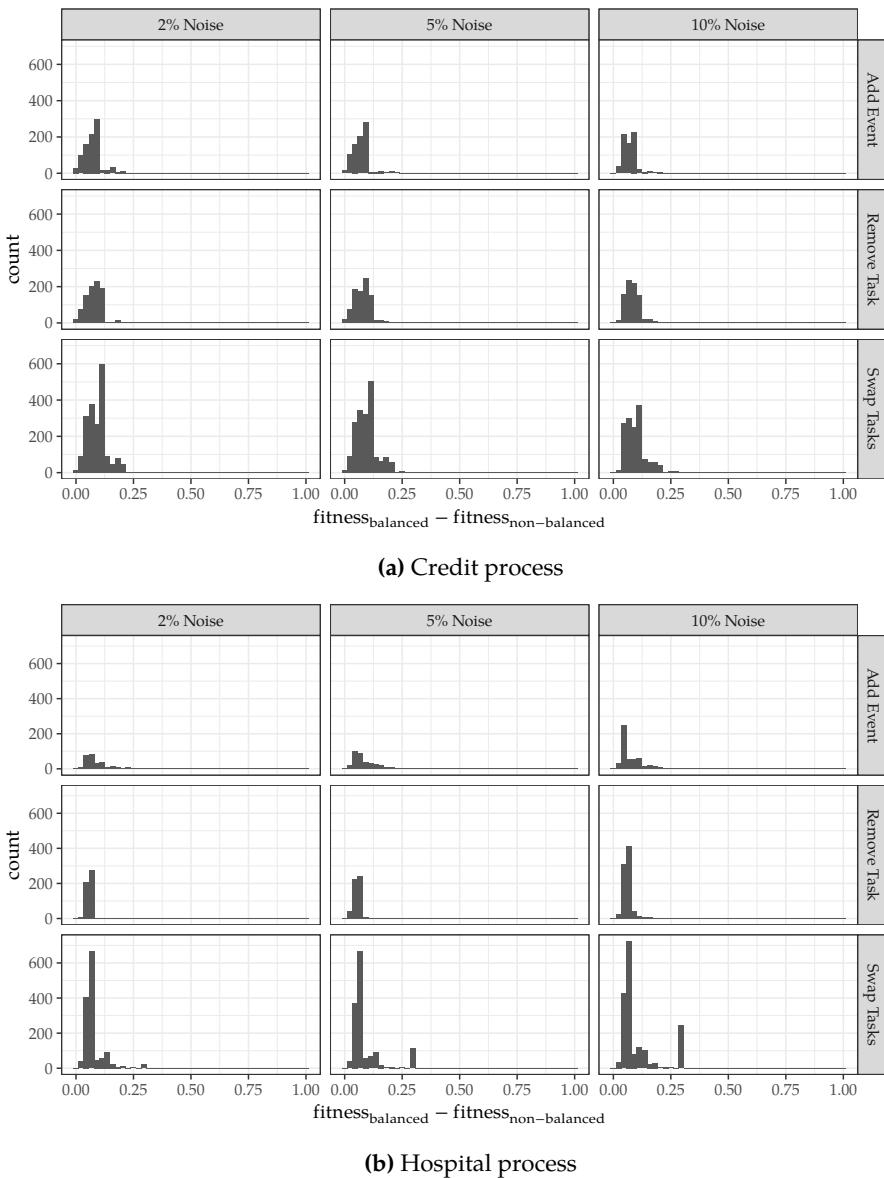


Figure 5.11: Absolute difference between the fitness determined by the balanced alignment method ($\text{fitness}_{\text{balanced}}$) and the non-balanced method ($\text{fitness}_{\text{non-balanced}}$). Note that the non-balanced method always returned a lower fitness compared to the balanced method.

we cannot expect any improvement in the fitness level for these traces. Among the remaining traces, several alignments that were returned by the non-balanced method [LA13a] returned as optimal were, in fact, suboptimal. Specifically:

- 24.6% of the alignments returned for the credit process and
- 18.6% of the alignments returned for the hospital process were suboptimal.

The difference between the fitness score returned by the non-balanced method and the balanced method was on average 0.09 for the credit process and 0.08 for the hospital process. This seems to be a small error. However, for several traces the non-balanced alignment was unable to compute an alignment, i. e., for

- 5.0% of the credit application process traces and
- 1.5% of the hospital process trace.

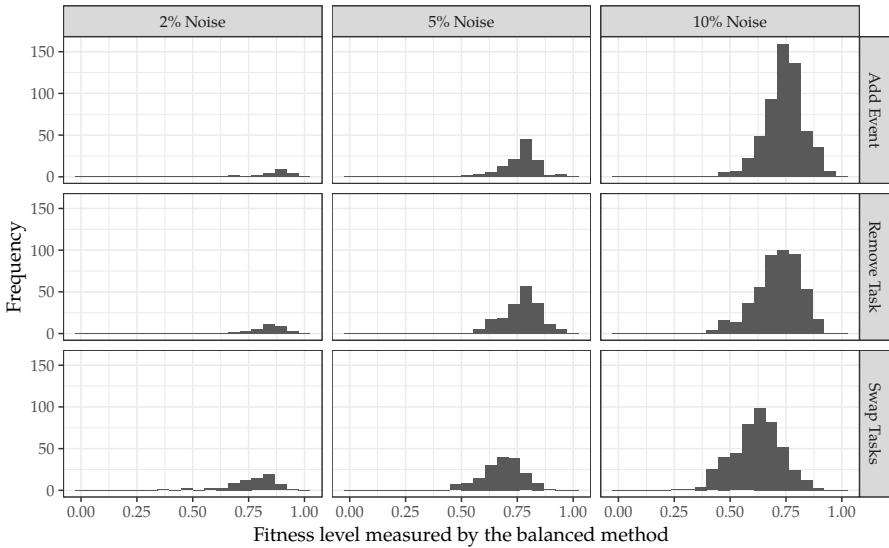
The non-balanced approach cannot not return an alignment when it is impossible to find a variable assignment for the sequence of transitions returned by the control-flow alignment. For example, it is impossible to find a variable assignment for the process trace

$$\langle (t_{tri}, w_1), (t_{ref}, w_2), (\tau_1, \emptyset), (t_{dia}, \emptyset), (t_{vis}, \emptyset), \\ (t_{dec}, w_3), (t_{pre}, w_4), (t_{org}, \emptyset), (t_{dis}, \emptyset) \rangle$$

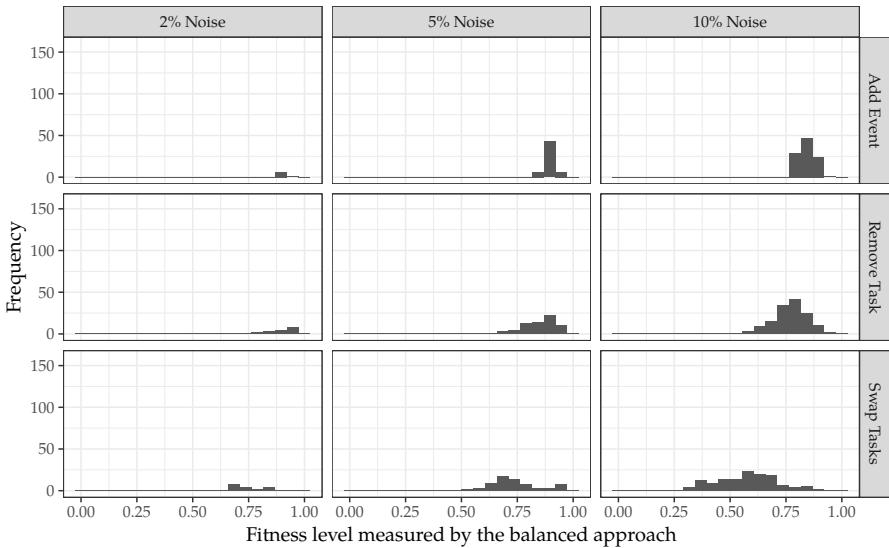
of the hospital process (cf., the DPN in Figure 3.3). There is no variable assignment for variable $data_{referral}$ in w_3 that fulfills the guards of both transitions t_{org} ($data_{referral} = \text{Tertiary}$) and t_{dis} ($data_{referral} = \text{Home}$). The non-balanced approach cannot change the sequence of transitions and fails to return an alignment.

Figures 5.11a and 5.11b show the distribution of the improvement in the fitness score obtained with the balanced method. Here, we only considered those traces for which the non-balanced approach did return an alignment. Among the improved alignments for the credit process, the average improvement of the fitness score is 0.09. The maximum improvement in fitness is 0.42. For the hospital process, the fitness score of the alignments improved by 0.08 on average. The maximum improvement was 0.30. Swapping tasks seems to increase the amount of traces for which the non-balanced approach returns sub-optimal alignments. However, increasing the noise level does not considerably increase the number of sub-optimal alignments as returned by the non-balanced approach.

Figures 5.12a and 5.12b show the distribution of the fitness score of traces for which the non-balanced approach did not return an alignment. For these traces, the average fitness was 0.76 for the credit process and 0.71 for the hospital process. The maximum fitness was, in both cases, as high as 0.96. Thus, the non-balanced approach could not compute an alignment even though alignments with very high fitness levels of up to 0.96 exist, i. e., some of these log traces deviated very little from the process traces prescribed by the model. Moreover, even for a low percentage of added noise (i. e., 2% of added, removed, or swapped events) the non-balanced approach cannot compute an alignment for some traces.



(a) Histograms of the fitness for the credit process



(b) Histograms of the fitness for the hospital process

Figure 5.12: Fitness level of traces determined by the balanced alignment method for which the non-balanced method [LA13a] could not compute an alignment.

5.4 Related Work

We first discuss related work on conformance checking that abstract from data-flow, resource constraints, and time constraints. Afterwards, we discuss related work that takes multiple perspectives into account.

5.4.1 Control-flow Conformance Checking

One of the earlier works that considers the problem of process conformance checking using event data is [CW99]. In [CW98a], the log is considered as a stream of events, which is matched to a model that is also considered as a stream of events. In contrast to our approach, no guarantees are made about the optimality of the result. In [BKR12] the conformance checking problem is tackled by defining the process as a Communicating Sequential Process (CSP) and applying a model checker to find counter examples. However, the method is limited to process models exhibiting finite behavior. In [Wei+11] and [RVA08], techniques are presented that compare an abstraction of a process model with a log. In both cases, the process model is required to exhibit finite behavior. Furthermore, no alignment is provided. Instead, only a number that quantifies the conformance is returned.

Token-based replay techniques [MAW08; MWA07; RA08; WAA06] can handle infinite behavior but need to resort to heuristics to deal with silent/duplicate activities. In some cases, the use of heuristics may lead to false negatives (i. e., perfectly fitting process executions are evaluated as non-fitting executions), as shown in [ADA11b]. Moreover, the user cannot set the severity of different deviations, i. e., non-conformance is measured in terms of missing and remaining tokens.

Efficient algorithms also exist to perform sequence alignments (e. g., the algorithms of Needleman-Wunsch [NW70] and Smith-Waterman [SW81]). Similarly, in process mining, Bose et al. [JA12] have proposed techniques to efficiently align pairs of log traces. Unfortunately, these approaches cannot be applied to find an alignment between a log trace and a process model. In our setting, the process trace that is to be aligned with the log trace is not known *a priori*; hence, a process trace minimizing the severity of the deviations needs to be chosen. Moreover, sequence and trace alignments only focus on the activity names ignoring the other perspectives.

To overcome the limitations of earlier approaches, such as no guarantees for correctness and the inability to handle silent/duplicate activities, alignment-based techniques were initially proposed by Adriansyah et al. [AAD12; ADA11b; Adr14]. These are tailored towards checking conformance between the control flow of a procedural process model and log traces. The A* algorithm together with a heuristic based on the Petri net marking equation has been proposed as an efficient solution to the problem [Adr14]. Indeed, our definition of a multi-perspective alignment is based on this seminal work. Further improvements to the control-flow alignment method have been proposed, e. g.:

- decomposing the problem in smaller problems [MCA14];
- translating alignment problems to constraint programming problems [L  p+16] or planning problems [LM17] and use off-the-shelf software to solve them;
- solving a series of Integer Linear Programming (ILP) problems that approximate the A* search [TC16];
- assuming a partially-ordered sequence as input [LFA15]; and
- using attribute data to construct a likely cost function [ALZ15].

Moreover, the alignment concept has also been transferred to declarative languages such as Declare [De +16; LMA12] and been applied directly to models in the BPMN notation [Mol+14]. Recently, there has been work on lifting the alignment between process models and event logs to a behavioral level rather than as an alignment between sequences [Gar+17]. Event structures are used as intermediate representation of both event log and process model. Then, behavioral differences are verbalized as natural language statements. One problem with such an approach is that it relies on a concurrency oracle to determine the event structure of the event log.

However, none of these extensions addresses the problem of aligning a multi-perspective process model to an event log. Unfortunately, the alignment technique proposed in [Adr14] and [Gar+17] cannot be straightforwardly extended to account for other perspectives such as time, resources, and data.

5.4.2 Multi-perspective Conformance Checking

Even though the major share of conformance checking work has been devoted to the control-flow perspective, there is work considering conformance of processes on multiple perspectives.

Rule-based conformance. There is related work in the area of compliance checking of business processes with regard to norms and regulations [ADW08; GMS06; HWG12; LMX07]. In contrast to our work, all these approaches focus on forward compliance checking, i. e., to verify whether a process model can exhibit non-compliant behavior by analyzing the model only, thereby ignoring event data. Our approach also notably advances beyond existing techniques for multi-perspective backward compliance checking [BLP12; Car13; CVB13b; Ly+11; Ram17]. All these compliance checking techniques verify rules on process behavior rule-by-rule and in isolation from the other constraints. Thus, such techniques only provide explanations for local compliance violations and do not provide a global explanation of deviating behavior based on a single process model for the overall process. These compliance techniques often use temporal logic representations such as LTL, for which the conformance can be verified efficiently. However, the overall conformance of the event log with regard to a given process model is not determined.

Model-based conformance. Other research approaches focus on verifying the compliance of process models with respect to declarative process models that are defined by a set of formulas, which are mostly intended to encode business rules of which one wants to verify the compliance (e. g., [BB14; BLP12; BMS16; CVB13b; DMM14; GGP15; Ly+11; Mon10]). For example, the work of Caron et al. [Car13; CVB13a; CVB13b] provides a comprehensive view on rule-based, multi-perspective compliance checking in the area of auditing and risk management. A log trace can be represented by a set of formulas (e. g., an event for activity A is followed by an event for activity B) and, hence, its compliance can be checked by applying existing techniques. Unfortunately, the diagnostics are limited to highlighting *which formulas* are not satisfied. To our knowledge, the same limitation is also shared by approaches that use alternative languages to handle verification with data variables in processes (e. g. [Alb+04]), as well as by techniques to debug the execution of distributed systems (e. g. [Rey+06; Xu+09]). The work by Borrego et al. [BB14] does provide some diagnostics on violated rules for data-aware declarative process models by using constraint programming. In [BB14] the minimum set of rules in the model that need to be relaxed in order satisfy all formulas is determined using constraint programming. However, the rules are limited to binary relationships between activities and violations in the event log, e. g., out of order events are not considered. Cook et al. [CHM01] extend their control-flow based event matching method [CW98a] to deal with timed models. However, the method does not find a globally optimal matching. Senderovich et al. [Sen+16a] consider the conformance between the execution of a stochastic process and a schedule. Here, the control-flow of the schedule is assumed to be conforming to the event log. However, we aim to pinpoint *where in the process* deviations occur, such as the case that an activity has not been executed or has written a wrong value for a variable. It is far from easy to derive the same insights on the basis of not-satisfied formulas. This is due to the fact that the same log trace can be *repaired* in multiple ways to satisfy one formula. When multiple, unsatisfied formulas come in to play, we would be interested in finding the least expensive changes that are needed to ensure all formulas are satisfied. In fact, this is again the problem of finding the least expensive solution in a certain search space, which is exactly what our application of the A^* algorithm aims to be.

Direct application of A^ .* In previous work [LAD12], de Leoni et al. have shown that, indeed, if the domains of all variables are finite, a multi-perspective alignment problem can be translated into the classical alignment problem. However, the assumption of finite domains for any variable is too restrictive and, hence, the practical relevance would be compromised. For example, guards could not be defined over variables with the domain of real numbers. If the domain of any variables is infinite, there are infinite successors to a given search space node, i. e., there are infinite alignments that can be obtained by adding an alignment

move. Therefore, the A* algorithm is not directly applicable because it requires the number of successors of a search space node to be finite. Our alignment method uses a technique that limits the number of successors to a finite number based on solving MILP problems.

Non-balanced approach. The balanced alignment method is significantly different from the approach proposed in [LA13a]. This approach performs the alignment computation in two steps. For each trace, a control-flow alignment is built leveraging work [Adr14]; then, the alignment is augmented with the write operations by solving a MILP problem. This approach is certainly faster since the A* algorithm only considers the control flow and one MILP problem needs to be solved in total. Unfortunately, in certain situations, the alignment is not optimal and, thus, can even return explanations of deviations that are unlikely from a domain viewpoint. By first considering only the control flow, this approach cannot balance the costs related to data and control-flow and, therefore, might return such wrong explanations as shown in Section 5.3. The technique proposed in our work is guaranteed to return optimal solutions and, hence, more likely explanations of diagnosed deviations. This is due to the fact that the different perspectives are considered all together rather than one-by-one.

5.5 Conclusion

In recent years, many techniques have been proposed to evaluate a model’s conformance with respect to given logs. As mentioned in Section 4.2, these techniques can only evaluate the conformance with respect to control-flow considerations. The techniques are unable to check, e. g., the correctness of routing decisions, activities performed by unqualified resources, and activities that happen outside the correct time frame.

5.5.1 Contribution

We present a method that computes an optimal, multi-perspective, balanced alignment. The computed alignment relates the behavior modeled in a multi-perspective process model with the behavior observed in an event log. The method balances deviations on the different process perspectives and provides an optimal explanation for the observed behavior in terms of an execution trace of the multi-perspective process model.

The main improvement over previous work presented in [LA13a] is that our approach allows to obtain an optimal solution for cost functions that cover all perspectives, thus, it is possible to give user-defined weight to the different perspectives. This allows us to express statements such as “Skipping activity *Check* is more severe than executing activity *Check* too late” and “Executing activity *Decide*

by a different doctor than activity *Visit* is less severe than sending patients with the triage color *Red* to their home". The method in [LA13a] return alignments that are not optimal in situations, in which control-flow violations are assigned a lower cost than violations in the data perspective. As a consequence, the explanation of the deviations may be unlikely or, even, wrong from a business viewpoint.

We tested the approach on several synthetic event logs to evaluate how the approach scales with traces of increasing length. Clearly, we cannot expect our method to be efficient in all cases since the worst case complexity of our method is exponential. Still, the experiments show that solutions for process models and event logs with real-life complexity can be found in a reasonable amount of time.

The presented balanced alignment method is fully implemented as plug-in *Conformance Checking of DPN* in the *DataAwareReplayer* package the open-source process mining framework ProM 6.7. The plug-in takes as input a process model in the form of a DPN and an event log in the XES format [IEECIS16]. It computes optimal balanced alignments for the traces of the event log. The MILP problems are solved through the open-source library *lpSolve*, which is based on the revised simplex method combined with a branch-and-bound method for the integers.¹¹ However, we use a standardized interface and, hence, it is easy to plug in other solvers like, e.g., Gurobi or CPLEX. We also tested the method with the commercial solver Gurobi. Finally, the Multi-perspective Process Explorer (MPE) tool, which we described in Section 11.2, is based on the balanced alignment method.

5.5.2 Limitations

We acknowledge that there are some limitations to our balanced alignment method.

- The flexibility of our balanced alignment method comes at a increased computational complexity of the alignment problem since the variable assignment is part of the search space. Compared to the non-balanced method [LA13a], we may need to solve a exponential number of MILP problems. We try to accommodate this increased complexity by several optimizations that are presented in Section 5.2.7.
- Defining a cost function for all perspectives may be difficult due to the complex interactions between the perspectives.
- Our method only returns one of several possible optimal alignments. Sometimes, there are multiple possible explanations with the same cost. In some situations it might be desirable to find all optimal alignments. Even though all optimal alignments are associated with the same cost, one of them might provide a better explanation from a domain viewpoint than another one.

¹¹<http://lpsolve.sourceforge.net>

- The applicability of our method relies on existing multi-perspective process model in which rules and regulations have been integrated. Such an integration of rules into one single process model can be challenging.

5.5.3 Future Work

There are several directions for interesting future work.

- It may be worth to explore whether other formulations of the search problem, e. g., using constraint programming or planning techniques, would lead to improvements to the computation time in practical cases. Some improvements have been reported when using constraint programming [L  p+16] or planning [LM17] to solve the control-flow alignment problem.
- Decomposition methods based on the concept of valid decompositions [Aal13] can be used to improve the performance of our multi-perspective conformance checking approach. An initial proposal for a valid decomposition of DPNs was made by de Leoni et al. in [Leo+14a]. The efficiency of these decomposition could be improved.
- Sometimes an optimal alignment is not required. An initial *good enough* explanation of the deviations between the observed events and the process model is sufficient to guide the search for conformance problems. Methods to quickly obtaining an approximate alignment with quality guarantees are needed. Some initial work on obtaining approximate control-flow alignments has been done in [Don+17; TC16], which might be extendable towards multi-perspective alignments. However, it is difficult to overcome the inherent complexity of the reachability problem, which needs to be solved in order to guarantee that the process projection of the alignment is a valid process trace in the model.
- Our method, as all other alignment methods, aligns log traces in isolation. In several scenarios, e.g. in process security checking, the conformance of a case depends on the behavior observed in other cases that are being executed. Aligning traces based on the whole log, i. e. a log-alignment, would not only compute an optimal alignment based on intra-case constraints, but a globally optimal alignment that considers inter-case constraints.
- Capturing the time perspective in constraint requires to encode the execution time of activities in extra variables and define guard expressions with complex calculations (cf., the constraint regarding the Check transition in Figure 3.3). Since time is monotonically increasing within a process instance, it would be possible to pre-compute some of the values in event attributes and simplify the guard expressions. An initial step towards this has been done in [Bal16].

6 Multi-perspective Precision

For the primary use cases of BPM, the discovered process model needs to *adequately* reflect the real behavior of the process. An obvious question is then: How does one know if a model is adequate? As discussed in Chapter 4, the quality of a process model can be measured along several quality dimensions. Clearly, a process model should be able to explain the behavior of the process using the process model: The model should *recall* the observed behavior. In other words, using process-mining terminology, the model should *fit* [Aal16] the real behavior observed in the event log. However, the model should also be *precise* [MC10]: It should not allow for more behavior than what was observed in the event log because the extra behavior would not have empirical support. Any extra behavior allowed by the model can be considered unlikely given the observed behavior.

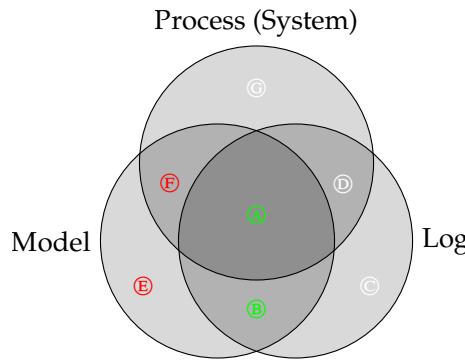


Figure 6.1: Venn diagram illustrating the precision measure [BDA14].

We can only observe the system behind the process through the event log. Therefore, the precision of a model is not an absolute value but, not different from fitness, it is relative to an event log (cf., Figure 6.1). In other words, the precision of a process model depends on what has been observed. We want to measure how much unobserved behavior is allowed by the process model. A process model is considered to be precise if the area $\textcircled{E} + \textcircled{F}$ in Figure 6.1 is small in comparison to the overall behavior allowed by the model ($\textcircled{A} + \textcircled{B} + \textcircled{C} + \textcircled{D}$). Note that for this definition of precision the original behavior of the process (system) is irrelevant. In fact, the boundary between area \textcircled{A} and \textcircled{B} as well as between \textcircled{E} and \textcircled{F} is generally unknown. Attempts to estimate the behavior of the original process (system) are

made to quantify the generalization dimension. However, as already motivated in Section 11.1.2, we do not consider the generalization dimension in this thesis.

Multiple methods to measure the precision of a process model based on event logs have been proposed in the literature [Adr+15; Bro+14; Gre+06; MC12; MWA07; RA08]. However, these methods can only be used to measure precision of models that do not encompass data-, resource, and time-related aspects. This is a serious limitation, since these aspects play an important role in real business processes. Ignoring those aspects hampers the applicability of the existing precision measures in practice: the extra constraints may exclude certain behavior and increase the precision of the model. Existing approaches are often unaware of multiple perspectives and, thus, tend to underestimate the precision of models.

6.1 Motivation for a Multi-perspective Precision Measure

We wish to illustrate the problem of measuring precision when ignoring perspectives beyond control flow. Let us consider a fragment of a credit application process that generated the event log L_c shown in Table 6.1. The event log contains six traces $\sigma_{c1}, \dots, \sigma_{c6}$ each of them containing four events. Two attributes are recorded: *resource*, the name of the person executing some of the activity and *loan*, the amount of the credit that is requested. The amount of the credit *loan* ranges from 750 to 5000 and each case is processed in a slightly different manner.

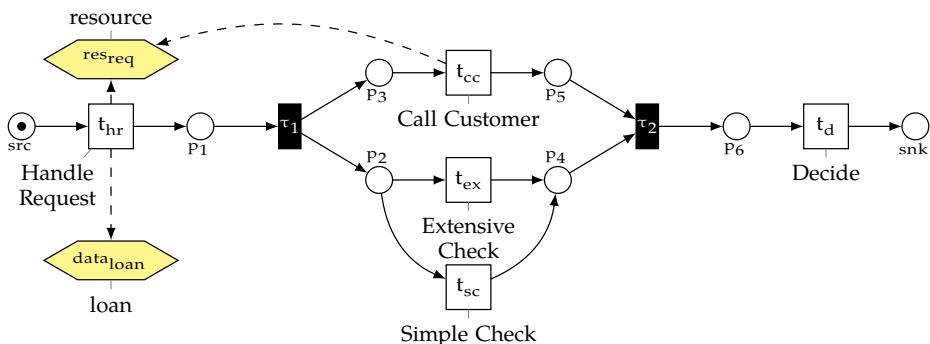
Figure 6.2 (N_1) and Figure 6.3 (N_2) show DPN models that describe the entire behavior of the process as it was recorded in L_c . Both the models are perfectly fitting with respect to the event log L_c . Moreover, when disregarding the data perspective model N_1 can be seen as a precise representation of the observed behavior.

The difference between models N_1 and N_2 is that the latter specifies additional rules: Depending on the requested loan amount either activity *Simple Check* or activity *Extensive Check* needs to be executed. For certain loan amounts, between 1,000 and 2,000, the decision between Simple Check ($data_{loan} < 2,000$) or Extensive Check ($data_{loan} > 1,000$) is left to the process worker. Moreover, in N_2 , a separation-of-duty constraint is implemented between activities *Call Customer* and *Handle Request*: These two activities must be performed by different resources (i. e. $res_{req} \neq res'_{req}$). Intuitively, these rules based on process data make the process model N_2 more precise than N_1 : Their presence provides additional constraints that reduce the amount of allowed behavior. For example, model N_1 would allow to execute *Simple Check* for any amount, but *Simple Check* is only considered for an amount smaller than 2,000 in the event log. The major limitation of all existing approaches [Adr+15; Bro+14; Gre+06; MC12; MWA07; RA08] is that they would return the same precision score for both models.

The main contribution of this chapter is a method that generalizes the precision measure proposed in [AAD12; Adr+15] to incorporate additional rules relating to multiple perspectives. More precisely, it supports multi-perspective rules that can

Table 6.1: Six traces of the event log L_c recorded by the example credit application process.

(a) Trace σ_{c1}				(b) Trace σ_{c2}				
id	activity	resource	loan	id	activity	resource	loan	
e _{c10}	Handle quest	Re-	Rory	750	e _{c20}	Handle quest	Rory	750
e _{c11}	Simple Check				e _{c21}	Call Customer	Amy	
e _{c12}	Call Customer	Amy			e _{c22}	Simple Check		
e _{c13}	Decide				e _{c23}	Decide		
(c) Trace σ_{c3}				(d) Trace σ_{c4}				
id	activity	resource	loan	id	activity	resource	loan	
e _{c30}	Handle quest	Re-	Rory	1250	e _{c40}	Handle quest	Rory	1500
e _{c31}	Simple Check				e _{c41}	Simple Check		
e _{c32}	Call Customer	Amy			e _{c42}	Call Customer	Amy	
e _{c33}	Decide				e _{c43}	Decide		
(e) Trace σ_{c5}				(f) Trace σ_{c5}				
id	activity	resource	loan	id	activity	resource	loan	
e _{c50}	Handle quest	Re-	Rory	1500	e _{c60}	Handle quest	Rory	5000
e _{c51}	Extensive Check				e _{c61}	Extensive Check		
e _{c52}	Call Customer	Amy			e _{c62}	Call Customer	Amy	
e _{c53}	Decide				e _{c63}	Decide		

**Figure 6.2:** Imprecise DPN model N_1 of the credit application process.

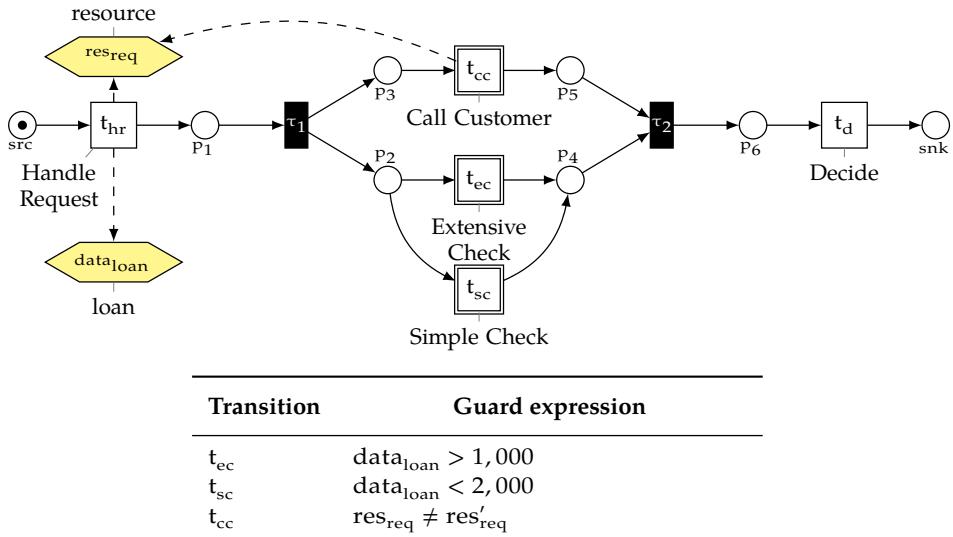


Figure 6.3: Precise DPN model N_2 of the credit application process that uses guards.

be encoded as constraints over data attributes and directly influence the execution of the process. The approach in [Adr+15] returns, for **both models, the same precision score** of 0.913, because it ignores the constraints of duty separation as well as those at decision points.

By contrast, our approach, when applied to the shown process model and event log, returns a **lower precision score** 0.757 for process model M_1 and a **higher precision score** 0.848 for the process model M_2 . Thus, the precision added by specifying data-driven rules for choices in process models is reflected in our measure. It fulfills the intuitiveness requirement [RA08]. Note that the values of the scores returned by our approach should not be compared directly to the scores returned by the approach in [Adr+15]. We compute the precision in a new, more generic manner that acknowledges the precision added by those rules and penalizes their absence. Therefore, both process models achieve lower precision scores. However, their relative ranking reflects the difference in precision. Moreover, the relative ranking between process models without multi-perspective rules remains the same as returned by the approach in [Adr+15].

6.2 Multi-perspective Precision Measure

We present our precision measure for multi-perspective process models. A large part of the work presented in this chapter was published in [Man+16d]. Before describing our method, we clarify the required input. As outlined in Figure 6.4, the

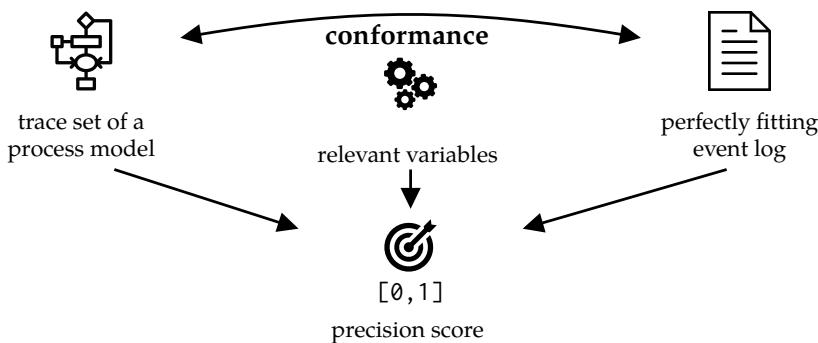


Figure 6.4: Overview of the proposed multi-perspective precision measure.

precision measure requires the trace set of a process model, an event log, and a set of attributes relevant to the precision measurement as input.

6.2.1 Assumptions on the Input

For the sake of a simpler presentation of our multi-perspective precision measure, we assume that the event log perfectly fits the process model, i.e., the fitness score defined in Section 4.4 returns a fitness score of 1.0 for the event log and the process model. A perfect fitness perfect fitness implies:

- that the control-flow is perfectly matching;
- that the names of log attributes and their values observed in the event log are matched with the ones in the process model; and
- that the event log contains events for unobservable routing activities (i.e., invisible transitions).

This does not limit the applicability of the approach since for any event log that does not meet these requirements, we can transform the log to the closest event log matching the requirements. For example, by using alignment-based techniques for multi-perspective process models as presented in Section 5.2. Using an alignment, we *squeeze* any non-compliant behavior present in the event log into the closest compliant behavior defined by the model. We add events for required unobservable activities, remove uncompliant events, and modify attribute values such that they fit the process model. Thus, we separate the fitness measurement from the precision measurement. Therefore, from now on, we assume that any event log has been preprocessed in such way.

6.2.2 Precision Measure

The *precision* of a process model in relation to an event log must take into account the extra behavior allowed by the model that is not seen in the event log. The *precision* of a process model is computed with respect to an event log that records executions of such a process. It is the ratio between the amount of observed behavior as recorded in the event log that is also described by the process model $\oplus + \otimes$ and the overall possible behavior as allowed by the process model $\oplus + \otimes + \circledast + \circledcirc$. All behavior that is allowed by the model yet never observed in the log makes a model less precise, i. e., we want to measure the size of the area $\circledast + \circledcirc$ in Figure 6.1.

To measure the amount of behavior that is allowed by the model yet never observed, we need a method to compare the possible behavior of the process with the observed behavior. Comparing the entire possible behavior of the process with the complete observed behavior in the event log is *not possible*, since we cannot assume that the, possibly, *infinite modeled behavior* can be observed in a *finite number of traces*. Several methods to approximate the preciseness of a process model with regard to an event log have been proposed [MC12; MWA07; RA08]. For our work, we adopt the idea proposed by Munoz-Gama et al. in [MC12]: *only the unobserved behavior that escapes from visited process states is considered*.

Process State

We generalize this idea towards multiple process perspectives. Therefore, we introduce the concept of a *process state*. We are only interested in process states that have been visited during the process execution that was recorded in the event log. As proposed for the control-flow perspective in [Adr+15], we use alignments to establish which part of the model has been used during the process execution.

We define a function that returns the *state of the process that is reached before an event occurred*. Given a log trace $\sigma \in \mathcal{E}$, we compute an optimal alignment and define function $state_{LTS,L}(e)$ such that it returns the prefix of process transitions of the process projection that is obtained just before the alignment move that contains event e . Moreover, we include the latest variable assignment in our notion of state. The variable values capture the current state of the other process perspectives (i. e., resources, time, and data).

Definition 6.1 (Prefix-based process state). Let V_P be a set of process variables with domain dom . Let $LTS = (TS, \lambda, \nu)$ be a labeled trace set. Let $L = (E, \Sigma, \#, \mathcal{E})$ be a perfectly fitting event log. Let $\sigma = \langle e_1, \dots, e_i, \dots, e_n \rangle$ be a log trace. Given an optimal alignment $\gamma_\sigma \in \Gamma_{L,LTS}^\sigma$, for each $e_i \in \sigma$ function $state_{LTS,L}(e_i) \in T^* \times U_{dom}^P$

is defined as follows:

$$state_{LTS,L}(e_i) = \begin{cases} (\langle \rangle, \emptyset) & \text{if } i = 1 \\ (\langle t_1, \dots, t_k \rangle, w) & \text{otherwise} \end{cases}$$

where

$$\begin{aligned} & \langle (\bar{e}_1, s_1), \dots, (\bar{e}_k, s_k), (e_i, s_{k+1}), \dots, (\bar{e}_n, s_n) \rangle = \gamma_\sigma \\ & \wedge proj_P(\langle (\bar{e}_1, s_1), \dots, (\bar{e}_k, s_k) \rangle) = \langle (t_1, w_1), \dots, (t_k, w_k) \rangle \\ & \wedge proj_L(\langle (\bar{e}_1, s_1), \dots, (\bar{e}_k, s_k) \rangle) = \langle e_1, \dots, e_{i-1} \rangle \\ & \wedge w_1 \oplus \dots \oplus w_k = w. \end{aligned}$$

◊

Thus, we can exactly describe the process state based on an event of the event log. Function $state_{LTS,L}$ is well-defined since only one optimal alignment is returned by the alignment method, e.g., Algorithm 1 can be implemented to deterministically return only one of the possible optimal alignments.

Several other state concepts are possible. There are many choices available and the choice of the state concept determines what constitutes precise and imprecise behavior. For example, we could use the multiset (or set) of activities that have been executed before event e was aligned (denoted as unordered alignment automaton by [Adr+15]). Moreover, we could also use the current marking and variable assignment of a DPN as state concept.

In this thesis, we only use the just introduced prefix-based state function. Using the prefix constitutes a strict notion of precision, e.g., for a process model with activities in two parallel branches all possible permutations of these activities need to have been observed to consider the model as precise. Other notions of process state could also be used together with our precision measure. Generally, we assume that the state notion should not be more coarse-grained than the real process state, e.g., using only the current marking of a DPN as the process state without considering the variable assignment would ignore the precision added by rules in the data perspective of the model.

Possible and Observed Behavior

The precision of a process model in relation to an event log must take into account the extra behavior allowed by the model that is not seen in the event log. We compute the precision of a process model with respect to an event log that records executions of such a process. It is the ratio between the amount of observed behavior as recorded in the log and the amount of possible behavior as allowed by the model. All behavior that is allowed by the model *yet never observed in the log* makes a model *less precise*.

More precisely, given an event log and a process model, we define the *possible behavior* that is possible in a state with respect to a set of process variables $V_{PR} \subseteq V_P$ and a particular event $e \in E$. We define the possible behavior based on all the

possible combinations of process transitions $t \in T$ and the values of the considered variables V_{PR} that can be executed in the process state prior to the occurrence of e . We allow to configure the considered variables V_{PR} depending on the use case. Often not all of them are relevant to measure precision.

Definition 6.2 (Possible Behavior). Let $LTS = (TS, \lambda, \nu)$ be a labeled trace set. Let $L = (E, \Sigma, \#, \mathcal{E})$ be a perfectly fitting event log. Let $state_{LTS,L}$ be a state function. Given a set of process variables $V_{PR} \subseteq V_P$ with domain dom , the possible behavior with respect to the variables V_{PR} that is allowed by the trace set LTS in the state before event e occurred can be represented as function $pos_{TM,L}(e, V_{PR}) \in T \times \mathcal{U}_{dom}^{PR}$. Function $pos_{TS,L}$ returns the *cartesian product of process transitions and variable assignments to variables V_{PR}* that are *possible according to LTS* when being in the state before event $e \in E$ occurred, i. e.:

$$\begin{aligned} pos_{LTS,L}(e, V_{PR}) = \{ & (t_{PR}, w_{PR}) \in T \times \mathcal{U}_{dom}^{PR} \mid \exists_{\sigma \in TS} \exists_{\langle (t_1, w_1), \dots, (t_n, w_n) \rangle \in PS^*} \\ & \langle (t_1, w_1), \dots, (t_n, w_n) \rangle \in prefix(\sigma) \\ & \wedge \langle (t_1, \dots, t_n), w_1 \oplus \dots \oplus w_n \rangle = state_{LTS,L}(e) \wedge t_n = t_{PR} \\ & \wedge \forall_{v \in V_{PR}} (w_n(v) = w_{PR}(v)) \}. \end{aligned} \quad \diamond$$

The definition above is kept intentionally generic concerning the choice of variables V_{PR} because we aim to measure the precision from different perspectives, which will be more extensively discussed in the following sections when we present two concrete multi-perspective precision measures. In a similar way, we define the *observed behavior* prior to the occurrence of any event $e \in E$ as all the combinations of process transitions and values of the considered variables V_{PR} that have been aligned to some event that occurred in the same state as prior to the occurrence of e .

Definition 6.3 (Observed Behavior). Let $LTS = (TS, \lambda, \nu)$ be a labeled trace set. Let $L = (E, \Sigma, \#, \mathcal{E})$ be an event log. Let $state_{LTS,L}$ be a state function. Given a set of process variables $V_{PR} \subseteq V_P$ with domain dom , we consider the observed behavior with respect to the variables V_{PR} that has been recorded according to the alignments between traces $\sigma \in \mathcal{E}$ and the labeled trace set LTS . The set of observed behavior can be represented as function $obs_{TS,L}(e, V_{PR}) \in T \times \mathcal{U}_{dom}^{PR}$. Function $obs_{TS,L}$ returns the *cartesian product of process transitions and assignments to variables V_{PR}* that have been *observed in the event log* when being in the state before event $e \in E$ occurred, i. e.:

$$\begin{aligned} obs_{LTS,L}(e, V_{PR}) = \{ & (t, w) \in (T \times \mathcal{U}_{dom}^{PR}) \mid \exists_{\bar{e} \in E} (state_{LTS,L}(e) = state_{LTS,L}(\bar{e})) \\ & \wedge \#_{activity}(e) = \lambda(m) \\ & \wedge \forall_{v \in V_{PR}} (\#(v(v))(e) = w(v)) \}. \end{aligned} \quad \diamond$$

Precision Measure

Using the definitions of possible and observed behavior in the context of an event, we define the precision of a trace set LTS according to variables $V_{PR} \subseteq V_P$ and an event log L as follows.

Definition 6.4 (Multi-perspective precision wrt. variables). Let $LTS = (TS, \lambda, \nu)$ be a labeled trace set. Let $L = (E, \Sigma, \#, \mathcal{E})$ be a perfectly fitting event log. Let $V_{PR} \subseteq V_P$ be a set of process variables. The precision of LTS with regard to L for the variables V_{PR} is a function $precision(LTS, L, V_{PR}) \in [0, 1]$:

$$precision(LTS, L, V_{PR}) = \frac{\sum_{e \in E} |obs_{LTS,L}(e, V_{PR})|}{\sum_{e \in E} |pos_{LTS,L}(e, V_{PR})|}. \quad \diamond$$

Note that we do not require the trace set trace set to be finite. We only consider the trace-set states that are visited by the recorded events $e \in E$. Since the number of events is finite, the number of considered process states is also finite. Furthermore, precision scores are always between 0 and 1 because for each event $e \in E$, all there is always at least as much possible behavior as observed behavior: $|obs_{LTS,L}(e, V_{PR})| \leq |pos_{LTS,L}(e, V_{PR})|$. This follows from our assumption made in Section 6.2.1. Clearly, all observed behavior needs to be possible according to the trace set for a perfectly fitting event log.

As discussed, most attention in the literature has been paid to computing the precision on the basis of activities that are enabled according to the model but never happened. This type of precision can be calculated as a special case of the precision measure defined in Definition 6.4 when setting $V_{PR} = \emptyset$, i. e., we consider only the possible process transitions as *escaping edges* and disregard the possible variable assignments. In the following, we will refer to this specific case as **activity precision**. Activity precision is concerned only with the possible behavior in terms of activities.

6.2.3 Activity-precision Measure

We present a specific application of the general precision measure by defining the multi-perspective *activity precision* of a process model, and proceed to show that our definition is intuitive by discussing a series of illustrative examples.

Definition 6.5 (Activity-precision measure). Let $LTS = (TS, \lambda, \nu)$ be a labeled trace set. Let $L = (E, \Sigma, \#, \mathcal{E})$ be a perfectly fitting event log. The activity precision of LTS with regard to L is a function $precision_{act}(LTS, L) \in [0, 1]$:

$$precision_{act}(LTS, L) = precision(LTS, L, \emptyset). \quad \diamond$$

The multi-perspective activity precision measure focuses on the ordering of activities, i. e., the control-flow. However, the effect of multi-perspective rules on the possible behavior is taken into account when calculating precision.

We proceed to show that our definition is intuitive by discussing a series of illustrative examples based on the event log of the credit application process and a collection of different process models N_1, N_2, N_3 , and N_4 . N_1 and N_2 are the process models introduced in Figure 6.2 and Figure 6.3. N_3 and N_4 are process models that represent extreme cases, which we will introduce later.

With abuse of notation, we assume that any DPN model N_i also refers to its trace-set representation. We obtain the following sets of observed and possible behavior for the events listed in Table 6.1 and the initial, imprecise model N_1 :

$$\begin{aligned} pos_{N_1, L_c}(e_{c10}, \emptyset) &= \{(t_{hr}, \emptyset)\}, & pos_{N_1, L_c}(e_{c11}, \emptyset) &= \{(t_{sc}, \emptyset), (t_{ec}, \emptyset), (t_{cc}, \emptyset)\}, \\ pos_{N_1, L_c}(e_{c12}, \emptyset) &= \{(t_{cc}, \emptyset)\}, & pos_{N_1, L_c}(e_{c13}, \emptyset) &= \{(t_d, \emptyset)\} \\ obs_{N_1, L_c}(e_{c11}, \emptyset) &= \{(t_{hr}, \emptyset)\}, & obs_{N_1, L_c}(e_{c12}, \emptyset) &= \{(t_{sc}, \emptyset), (t_{cc}, \emptyset)\}, \\ obs_{N_1, L_c}(e_{c13}, \emptyset) &= \{(t_{cc}, \emptyset)\}, & obs_{N_1, L_c}(e_{c14}, \emptyset) &= \{(t_d, \emptyset)\} \end{aligned}$$

For example, the set of observed behavior for e_{c11} is $\{(t_{sc}, \emptyset), (t_{cc}, \emptyset)\}$ because the execution of both activities *Simple Check* and *Call Customer* can be observed in those events that are recorded when the process is in the same state as prior to the occurrence of e_{c11} : $state_{N_1, L_c}(e_{c11}) = \langle (t_{hr}), (res_{req} \mapsto \text{Rory}, data_{load} \mapsto 750) \rangle$. This state is reached when activity *Handle Request* has already been executed and the latest values assigned to the attributes *Resource* and *Loan* are *Rory* and 750 respectively.

By consulting Table 6.1, it becomes clear that events e_{c11} and e_{c21} contribute to the set of observed behavior for e_{c11} . Please note that e_{c11} and e_{c21} are events from different traces, i. e., the whole event log is considered when computing the observed behavior. Continuing in the same manner with the remaining events yields $precision(N_1, L_c) = \frac{28}{37} \approx 0.76$.

Applying the same measure on the process model N_2 , we get:

$$\begin{aligned} pos_{N_1, L_c}(e_{c10}, \emptyset) &= \{(t_{hr}, \emptyset)\}, & pos_{N_1, L_c}(e_{c11}, \emptyset) &= \{(t_{sc}, \emptyset), (t_{cc}, \emptyset)\}, \\ pos_{N_1, L_c}(e_{c12}, \emptyset) &= \{(t_{cc}, \emptyset)\}, & pos_{N_1, L_c}(e_{c13}, \emptyset) &= \{(t_d, \emptyset)\} \\ obs_{N_1, L_c}(e_{c11}, \emptyset) &= \{(t_{hr}, \emptyset)\}, & obs_{N_1, L_c}(e_{c12}, \emptyset) &= \{(t_{sc}, \emptyset), (t_{cc}, \emptyset)\}, \\ obs_{N_1, L_c}(e_{c13}, \emptyset) &= \{(t_{cc}, \emptyset)\}, & obs_{N_1, L_c}(e_{c14}, \emptyset) &= \{(t_d, \emptyset)\}. \end{aligned}$$

This results in a value of $precision(N_2, L_c) = \frac{28}{33} = 0.848$. It is easy to see that the added constraints regarding the attribute *Loan* limit the set of possible activities for event e_{c11} to *Simple Check* and *Call Customer*. The activity *Extended Check* cannot be executed anymore in the state prior to the occurrence of e_{c11} , as the value of the attribute *Loan* would need to be higher than 1,000. This improves the multi-perspective activity precision of model N_2 .

Moreover, the observed parallelism of activities *Simple Check* and *Call Customer* for a *Loan* value of 750 is not seen as imprecision in either case, as reflected in the set of observed behavior $\{(t_{sc}, \emptyset), (t_{cc}, \emptyset)\}$ for event e_{c11} . Note that the assignment of data attributes needs to be **exactly** the same in order to detect parallelism in the model as a precise representation of the observed behavior. Otherwise, when parallelism is observed with different attribute values, it is seen as an imprecision because a more precise process model using the different attribute values for a data rule can be created.

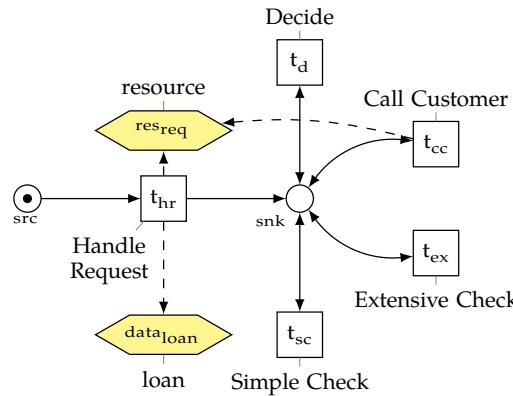


Figure 6.5: Very imprecise DPN model N_3 of the credit application process.

Next to the two process models introduced in Figure 6.2 and Figure 6.3, we consider two additional, illustrative process models representing extreme cases. Figure 6.5 shows an example of a process model that is very imprecise in relation to event log L_c . Always starting with *Handle Request*, model N_3 in Figure 6.5 allows to execute the remaining activities any arbitrary number of times and, also, in any order. On the opposite side of the spectrum, model N_4 in Figure 6.6 is very precise, as only exactly the observed behavior in L_c is possible. For instance, the order of the activities *Simple Check* and *Call Customer* depends on the value of the data attribute *Loan*; if it has 750 as value both activities are carried out in any order, for the values 1,500 and 1,250 only the modeled order is observed.

For process model N_3 , we obtain $precision(N_3, L_c) = \frac{28}{78} \approx 0.359$, which is less than half of the precision measure of N_2 in Figure 6.3 (0.848). On the other end of the spectrum, model N_4 in Figure 6.6 has a perfect precision: $precision(N_4, L_c) = 1$, as only the behavior seen in the event log is allowed. These examples demonstrate that the computed precision values behave as expected. Model N_3 is, indeed, the least precise, and model N_4 is the most precise of the presented examples. Models scoring a very high precision value are not always the most preferable. In particular, model N_4 allows for exactly the behavior observed in the event log and nothing

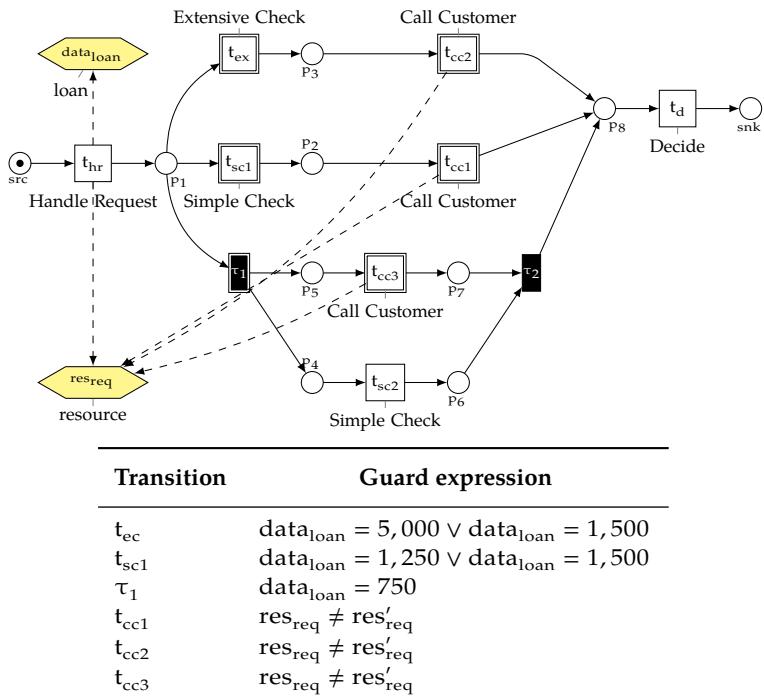


Figure 6.6: Perfectly precise DPN model N_4 of the credit application process.

more. Given that event logs only contain example behavior as observed in a limited time frame, one cannot assume that all possible behavior has been observed. In other words, model N_4 is probably *over-fitting* the event log.

6.2.4 Resource-precision Measure

Our multi-perspective activity-precision measure determines the unnecessary behavior defined by a process model in terms of enabled activities. We denote it as activity-precision measure since only the possible process transitions are considered as sources of imprecision. However, multi-perspective process models also encompass behavior in the time-, resource-, and data-perspective, i. e., several resource may execute an activity or several values may be assigned to a data attribute. Here, we show that it is also possible to use our general precision measure for measuring the precision of those multi-perspective assignments.

Generally, our measure is directly applicable to any kind of variable that is part of the process model and recorded as an attribute in the event log. The idea is to measure how many of the possible variable assignments are observed in the event log. As an example of the general idea, we sketch how to use our measure to determine the precision of the *resource* perspective of process models. The same idea could be directly applied to any other nominal variable¹², e. g., customer categories (bronze, silver, and gold). We assume that information about the resources working in a process can be encoded in the data attribute *resource*, which is defined over a finite domain, i. e., there is a finite number of resources taking part in the process. This makes it is possible to measure the *resource precision*. Given the sets of resources in a process (e. g., groups of employees in a certain roles), and constraints governing the allocation of activities to those sets, we can measure the precision of the allocation.

For example, consider the models in N_1 (Figure 6.2) and N_2 (Figure 6.2). We assume that in both models a limited set of resources $\{Amy, Rory\}$ can execute activities. Clearly, the separation-of-duty constraint (i. e., those activities must not be executed by the same person) in N_2 increases the precision of the resource allocation. In fact, it limits the resources that are allowed to perform *Call Customer*, i. e., given that *Rory* performed *Handle Request*, only *Amy* is left to perform *Call Customer*. To apply the generic precision measure on the resource perspective, we need to consider the possible resources together with the possible activities.

Definition 6.6 (Resource-precision measure). Let $LTS = (TS, \lambda, \nu)$ be a labeled trace set. Let $L = (E, \Sigma, \#, \mathcal{E})$ be a perfectly fitting event log. The resource precision of LTS with regard to L is a function $precision_{res}(LTS, L) \in [0, 1]$:

$$precision_{res}(LTS, L) = precision(LTS, L, \{\text{resource}\}). \quad \diamond$$

¹²Discrete and continuous variable would require special pre-processing steps since it is unlikely that all values of such variable are observed. We discuss this case as future work in Section 6.6.3

Thus, the generic multi-perspective precision measure can be used to determine how precisely variables with a finite domain, such as resources, are assigned. In the scope of this thesis, we only present the application of our measure on the resource-perspective. Resource precision is only an example of an application of our generic precision measure for multi-perspective process models. A similar measure could also be directly applied to any other process variables with a finite domain, e.g., categories of customers and the triage priority or the referral type assigned to patients (cf., Figure 3.3).

Again, we proceed to show that our definition of resource-precision is intuitive by discussing it using two examples models N_1 and N_2 in Figures 6.2 and 6.3. We define $V_r = \{\text{resource}\}$ and obtain the following sets of observed and possible behavior for model N_2 :

$$\begin{aligned} pos_{N_2, L_c}(e_{c10}, V_r) &= \{(t_{hr}, (\text{res}_{\text{req}} \mapsto \text{Amy})), (t_{hr}, (\text{res}_{\text{req}} \mapsto \text{Rory}))\}, \\ pos_{N_2, L_c}(e_{c11}, V_r) &= \{(t_{sc}, \emptyset), (t_{cc}, \emptyset)\}, \\ pos_{N_2, L_c}(e_{c12}, V_r) &= \{(t_{cc}, (\text{res}_{\text{req}} \mapsto \text{Amy}))\}, \\ pos_{N_2, L_c}(e_{c13}, V_r) &= \{(t_d, \emptyset)\} \\ obs_{N_2, L_c}(e_{c11}, V_r) &= \{(t_{hr}, (t_{hr}, (\text{res}_{\text{req}} \mapsto \text{Rory})))\}, \\ obs_{N_2, L_c}(e_{c12}, V_r) &= \{(t_{sc}, \emptyset), (t_{cc}, \emptyset)\}, \\ obs_{N_2, L_c}(e_{c13}, V_r) &= \{(t_{cc}, (\text{res}_{\text{req}} \mapsto \text{Amy}))\}, \\ obs_{N_2, L_c}(e_{c14}, V_r) &= \{(t_d, \emptyset)\} \end{aligned}$$

The separation-of-duty constraint in N_2 limits the possible resources that may execute activity *Call Customer*. We obtain $pos_{N_2, L_c}(e_{c12}, V_r) = (t_{cc}, (\text{res}_{\text{req}} \mapsto \text{Amy}))$. This results in a resource precision of $\frac{28}{55} \approx 0.509$. Model N_1 does not define such a constraint and, thus, we obtain the following sets of observed and possible behavior for model N_1 :

$$\begin{aligned} pos_{N_1, L_c}(e_{c10}, V_r) &= \{(t_{hr}, (\text{res}_{\text{req}} \mapsto \text{Amy})), (t_{hr}, (\text{res}_{\text{req}} \mapsto \text{Rory}))\}, \\ pos_{N_1, L_c}(e_{c11}, V_r) &= \{(t_{sc}, \emptyset), (t_{cc}, \emptyset)\}, \\ pos_{N_1, L_c}(e_{c12}, V_r) &= \{(t_{cc}, (\text{res}_{\text{req}} \mapsto \text{Amy})), (t_{cc}, (\text{res}_{\text{req}} \mapsto \text{Rory}))\}, \\ pos_{N_1, L_c}(e_{c13}, V_r) &= \{(t_d, \emptyset)\} \\ obs_{N_1, L_c}(e_{c11}, V_r) &= \{(t_{hr}, (t_{hr}, (\text{res}_{\text{req}} \mapsto \text{Rory})))\}, \\ obs_{N_1, L_c}(e_{c12}, V_r) &= \{(t_{sc}, \emptyset), (t_{cc}, \emptyset)\}, \\ obs_{N_1, L_c}(e_{c13}, V_r) &= \{(t_{cc}, (\text{res}_{\text{req}} \mapsto \text{Amy}))\}, \\ obs_{N_1, L_c}(e_{c14}, V_r) &= \{(t_d, \emptyset)\} \end{aligned}$$

Due to the missing separation-of-duty constraint on transition t_{cc} , the possible behavior before event e_{c12} occurred is $\{(t_{cc}, (\text{res}_{\text{req}} \mapsto \text{Amy})), (t_{cc}, (\text{res}_{\text{req}} \mapsto \text{Rory}))\}$. Both *Amy* and *Rory* may execute the *Call Customer* activity. However, only *Amy* is

observed to execute this activity. Therefore, N_1 gets a lower resource-precision than N_2 , its resource precision is $\frac{28}{74} \approx 0.378$. This follows the intuition that a model that defines constraints on the resource perspective should obtain a higher precision score than a model without such constraints.

Again, similar to the discussion about the activity-precision score of a model, a high resource-precision score should not be the only objective to measure the quality of a process model. However, a good process model should strike a balance between fitness and precision.

6.3 Locating Precision Problems

So far, we have only discussed global measures of precision (both activity and resource) for the entire process model. However, it is often desirable to pinpoint the sources for imprecision in a process model. Being aware of the imprecise model parts it is, e. g., possible to use decision mining techniques to discover data rules that increase the precision of a model in certain parts. We introduce such a decision mining technique later in Chapter 10.

To better evaluate how the addition of a single data rule affects the precision of a process model, we introduce a local variant of the precision measure. The idea behind this measure is that only the behavior local to a decision point is considered when computing the sets of possible and observed behavior. Non-local behavior, e. g., in different parallel branches, is not considered since data rules that are local to a decision point cannot influence behavior in parallel branches. Our goal is to visualize the local precision on the process model, thus, we use DPN as a concrete modeling notation instead of trace sets for its definition. We define the following refinements of the possible and observed behavior and a local place-precision measure.

Definition 6.7 (Locally possible- and observed behavior). Let $LTS = (TS_{N,M_I,M_F}, \lambda, \nu)$ be the labeled trace set induced by the labeled DPN $LN = (N, \lambda, \nu)$ with initial marking M_I and final marking M_F . Let $L = (E, \Sigma, \#, \mathcal{E})$ be a perfectly fitting event log. Let $p \in P$ be a place of LN . Let $V_{PR} \subseteq V_P$ be a set of process variables. Let $e \in E$ be an event. Function $pos_{LTS,L,p}(e, V_{PR}) \in T \times \mathcal{U}_{dom}^{PR}$ returns local behavior with respect to the variables V_{PR} that is possible for output transitions of place p , i. e.:

$$pos_{LTS,L,p}(e, V_{PR}) = \{(t, w) \in pos_{LTS,L}(e, V_{PR}) \mid t \in p\bullet\}$$

Function $obs_{LTS,L,p}(e, V_{PR}) \in T \times \mathcal{U}_{dom}^{PR}$ returns local behavior with respect to the variables V_{PR} that is observed for output transitions of place p , i. e.:

$$obs_{LTS,L,p}(e, V_{PR}) = \{(t, w) \in obs_{LTS,L}(e, V_{PR}) \mid t \in p\bullet\}$$

◊

Definition 6.8 (Local precision measure). Let $LTS = (TS_{N,M_I,M_F}, \lambda, \nu)$ be the labeled trace set induced by the labeled DPN $LN = (N, \lambda, \nu)$ with initial marking M_I and final marking M_F . Let $L = (E, \Sigma, \#, \mathcal{E})$ be a perfectly fitting event log. Given a set of process variables $V_{PR} \subseteq V_P$ and a place $p \in P$ of LN , the local precision of place p with regard to L for the variables V_{PR} is a function $localPrecision(LTS, L, V_{PR}, p) \in [0, 1]$:

$$localPrecision(LTS, L, V_{PR}, p) = \frac{\sum_{e \in E} |obs_{LTS,L,p}(e, V_{PR})|}{\sum_{e \in E} |pos_{LTS,L,p}(e, V_{PR})|}. \quad \diamond$$

In Figures 6.7 and 6.8 we projected the local activity-precision measure on two examples DPN models of the credit application process. We color coded the precision result such that a dark color represents a very low precision score and a bright color represents a good precision score. Except for place p_2 , all other places have a perfect local precision score of 1.0. This is because those places have only one output transition, i. e., $|p \bullet|$. It follows that only one transition is in the set of possible and observed local behavior and, thus, the local precision of those places is 1.0. The local precision measure ignores imprecision that are caused due to parallelism.

As discussed earlier, the difference between models N_1 and N_2 is the added guard expression in N_2 for both output transitions t_{ec} and t_{sc} of p_2 . This influences the local precision at p_2 . The place obtains a precision score of 0.643 for N_1 and a local precision of 0.9 for N_2 . The color coding of local precision values is implemented in the Multi-perspective Process Explorer (MPE) tool, which is introduced in Section 11.2 and used in several case studies. Moreover, we use the local precision measure along with the alignment-based fitness measure to evaluate the decision mining method presented in Section 10.3.

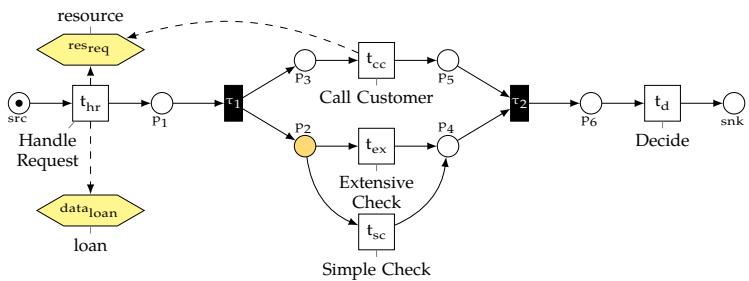


Figure 6.7: Local precision measure projected on the places of the imprecise DPN model N_1 .

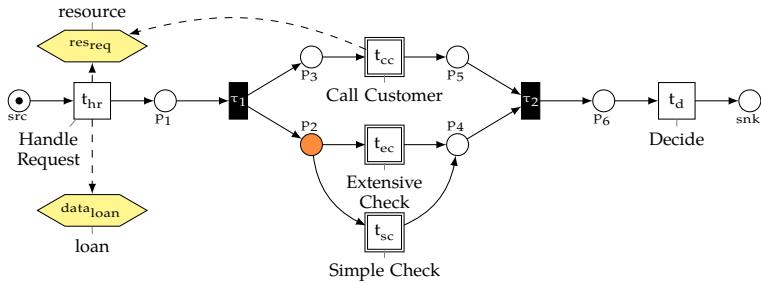


Figure 6.8: Local precision measure projected on the places of the more precise model N_2 .

6.4 Evaluation

The evaluation is based on a real-life case study, involving the processing of road-traffic fines by an Italian local police force [Man+16d]. Specifically, we employed a real-life event log [LM15] that records the execution of 11 different activities and contains 550,000 events grouped into 150,000 traces. In total there are 9 data attributes.

For this evaluation, we used five different DPN process models:

Model A, which is discovered using the Inductive Miner set to guarantee perfect fitness [LFA13];

Model B, which extends model A with guards as discovered by the decision-tree miner [LA13b]; *the minimal instances per decision-tree leaf* parameter was set to 125 to avoid over-fitting;

Model C, which is the normative model, shown later on in Figure 12.1a and first introduced by [Man+16a], but without any guards;

Model D, the normative model from Figure 12.1a again, yet including all those guards that concern attributes available in the public event log;

Model E, which extends Model C with the guards discovered with the decision tree miner (using the same parameter settings as for model B).¹³

Table 6.2 shows the precision and fitness scores for the described process models. We used the just described activity-precision measure and the multi-perspective fitness measure described in Section 5.2. We do not report on the difference between the multi-perspective precision values and the existing precision measures, since it is clear that both measures are incompatible. We already showed in Section 6.1 that existing methods would ignore the presence of data rules and do return the same precision score for models A and B, as well as for models C, D, and E. Clearly, the precision score of these models should not be identical since the added data rules contribute towards to precision of the process models. Our multi-perspective preci-

¹³All used models can be downloaded from <http://purl.tue.nl/726309911741849>

Table 6.2: Precision and fitness scores for the normative and discovered process models of a process managing road traffic fines enacted in an Italian municipality.

Process Model	Precision	Fitness
A: Inductive Miner	0.298	1
B: Inductive Miner with discovered rules	0.344	0.922
C: Normative model without rules	0.639	0.997
D: Normative model	0.694	0.972
E: Normative model with discovered rules	0.801	0.981

sion measure is incompatible with existing measure since it needs to assign a lower precision to process models without data rules whereas existing measures simply ignore the presence of data rules. As both existing measures and our proposed measure normalize values on a scale from 0 to 1, their result cannot be directly compared. In the remainder of this evaluation, we show that our precision measure results in intuitive scores when comparing process models without data rules to process models with discovered data rules.

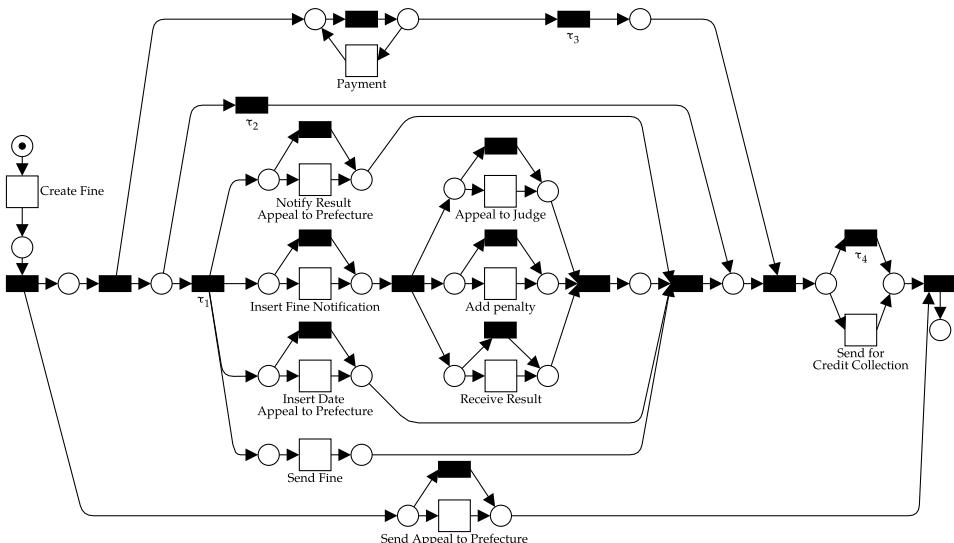


Figure 6.9: Model A discovered by the Inductive Miner on the road-traffic fine event log.

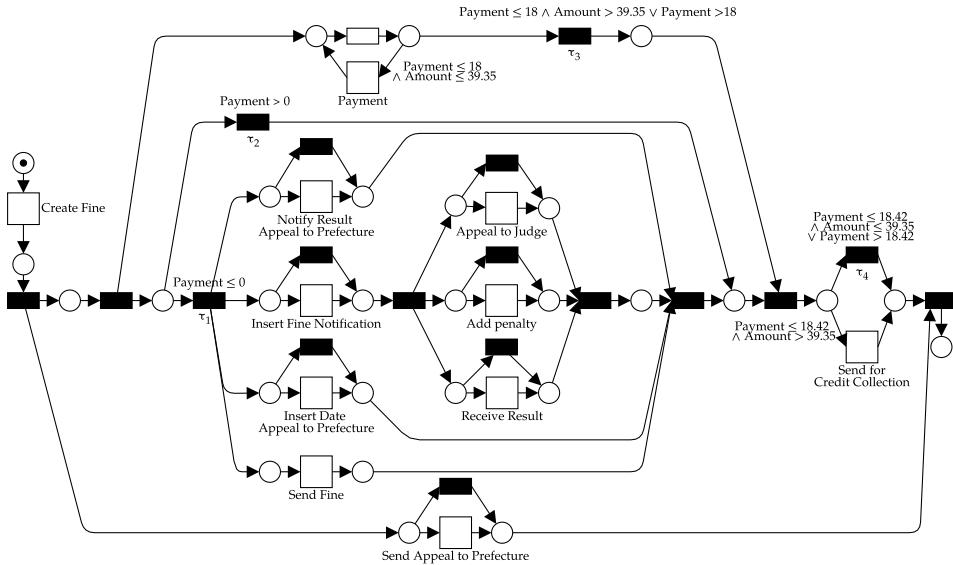


Figure 6.10: Model B based on the model A with data rules discovered by using the ProM decision miner. Write operations are omitted to improve the readability on paper.

Models A and B. Intuitively, model A should be the least precise process model. This model does not constrain the allowed behavior with any data rules. Also, the Inductive Miner, with parameters that are set to guarantee perfect fitness, is unlikely to discover a precise model for this event log, which includes infrequent behavior. Indeed, Model A scores a low precision of 0.298. Figure 6.10 shows model B, which has the same control-flow as model A, but additional guards based on the discovered rules. As expected, the discovered rules in model B result in an improved precision of 0.344. This comes at the expense of a slightly lower fitness score. Still, model B arguably allows for too much behavior, which is reflected in the poor precision score.

Models C, D, and E. The normative model without data rules, model C, is more precise than the models discovered with the Inductive Miner; its precision is 0.639. As expected, adding the normative data rules shown in Figure 5.7 to arrive at model D will increase its precision to 0.694. However, adding those data rules has an impact on the fitness of model D; its fitness is reduced by 0.025. As reported in [Man+16a], the event log shows that the rules are not always respected. Finally, we applied the DTM on the normative model, which resulted in model E. It scores best on precision with a score of 0.801, and better on fitness than model D. Completely in line with expectations, Model E scores better in fitness, because it discovers the as-is rules

rather than the to-be rules. A cursory glance on the results may invoke surprise that the precision of model E is higher than the precision of model D. However, this can be expected: The guards are discovered so as to maximize fitness and precision. Therefore, the rules added in model D only reflect constraints on the process from a compliance perspective, e.g. *Send of Credit Collection* should only be executed if the fine is not yet fully paid. By contrast, the decision tree miner strives for discovery of mutually-exclusive rules that describe the real behavior as observed in the event log. Being based on the real process executions, these rules may violate the normative rules. There are mutually exclusive rules discovered for both transitions *Send Fine*: $\text{Payment} > 18$ and τ_1 : $\text{Payment} \leq 18$, which leads to a higher precision score than the rules defined for the normative model D. In model D only a data rule for transition τ_1 is specified: $(\text{Dismissal} \neq \text{NIL} \vee (\text{Points} = 0 \wedge \text{Payment} \geq \text{Amount}))$. These controlled experiments using real-life data show that our way of computing precision is applicable to evaluate the quality of multi-perspective process models and provides intuitive results.

Regarding the computational complexity of the approach, we conducted all experiments on a standard laptop with a memory limit of 2 GB, which is lower than what current-day, regular computers contain. Given a perfectly fitting event log, the precision measurement could be computed within at most 12s for all considered process models. However, creating a perfectly fitting event log requires to compute an alignment. We reported extensively on the performance of multi-perspective alignments in Section 5.3. Therefore, since the complexity of the precision measurement is dominated by the alignment computation, we do not repeat performance experiments here.

6.5 Related Work

Measuring the precision of a process model is *not trivial*. Process models may specify an *infinite* amount of behavior. Therefore, it is not possible to check if all possible behavior is actually observed in the event log, since it is impossible that infinite behavior can be observed in a finite number of traces.

Multiple measures for *precision* have been proposed in the literature [AAD12; Adr+15; Bro14; Gre+06; MC12; MWA07; RA08] that extrapolate a finite amount of significant process behavior out of such in-finiteness. The work presented in [Gre+06; MWA07; RA08] introduced various measures for precision, all based on comparing observed and modeled behavior: soundness [Gre+06]; behavioral precision [MWA07]; and advanced behavioral appropriateness [RA08].

A fundamentally different approach to determine precision is proposed in [Bro14; Wee+11]. Both approaches insert artificially generated negative events into the event log, and determine the precision using the standard precision measure from binary classification. Similarly, in work [DCC16] highly deviating model traces with respect to the event log are considered to determine the precision and generalization of a

model. In [MC12], another precision measure is proposed, which counts so-called *escaping edges* while traversing the process models behavior based on the event log. Works [AAD12; Adr+15] extend this approach by using alignments between observed and modeled behavior to cope with event logs that deviate from the modeled behavior.

Our approach shares several similarities with the research reported on in [AAD12; Adr+15]. However, there are two major differences between all existing measures and our proposed approach:

1. We consider the influence of data rules on the behavior of a process model when calculating the precision score.
2. We generalize the precision measure beyond the limited scope of the control-flow perspective enabling to determine other types of precision, e. g., precision of resource assignments.

None of the mentioned works consider the importance of data-driven rules constraining the behavior possible according to a process model. Data-driven rules involving process data are readily available in most process modeling languages such as, e. g., data-driven gateways in BPMN. Moreover, they are used widely in practice, e. g., as illustrated by the recent uptake of the DMN standard [DMN16] in industry.

6.6 Conclusion

Whereas process modeling languages commonly used in practice (e. g. BPMN and DMN) allow one to specify data-driven rules to model choices, existing approaches to measure the precision of a process model ignore perspectives other than the control-flow perspective. The precision of a process model can be seen as the fraction of the possible behavior allowed by the model in relation to what has actually been observed, as recorded in the event log.

6.6.1 Contribution

In this chapter, we proposed a new measure for the **precision of multi-perspective process models** in relation to behavior that is described in the form of an event log. In particular, given an event log of a certain process and a number of models for the same process, we are able to determine which model scores higher on precision.

The chapter is based on our publication [Man+16d], which reports on the *first proposal* to measure precision for multi-perspective process models. We assume that information about process perspectives such as resources, time and data are recorded in the attributes the event log and that multi-perspective constraints are defined with rules over such data. The proposed precision measure generalizes

existing precision measures [MC12] by taking these rules and the recorded data values into account. The proposed measure is configurable towards multiple concepts of precision. Closest to the existing work on precision measures is the presented activity-precision measure. This measure considers the number of enabled activities in any given state as the source of imprecision. Next to this more traditional view on precision, we also presented the resource-precision measure, which also considers the set of available resources for activities as a source of imprecision.

For each model, the respective precision score can be combined with the fitness score, which, for instance, can be computed using the balanced alignment method described in the previous Chapter 5. In many cases, higher values of precision are associated with lower values of fitness, and vice versa. By finding the right trade-off between these two quantities, we can determine which model provides a better representation of the process in question.

Both the activity-precision and the resource-precision measure have been implemented as plug-ins of ProM 6.7 in the *DataAwareReplayer* package. Moreover, in Section 11.2, we introduce the Mulit-perspective Process Explorer, an interactive tool, that uses the activity-precision measure to determine the quality of discovered data rules and visualizes the proposed local place-precision measure.

6.6.2 Limitations

We acknowledge that there are some limitations to our multi-perspective precision measure.

- We only estimate the precision of the process model with regard to the behavior observed in the event log. Our method does *not measure* the precision of the model with regard to the underlying system that generated the event log. In the Venn diagram shown in Figure 6.1 our method considers the area \oplus as imprecision and the area \ominus as precise representation whereas the behavior in area \oplus correctly describes the underlying system and the behavior in area \ominus is not part of the system behavior. We take the position that the measurement of the model precision with regard to the system is a different and very challenging problem orthogonal to the measurement of the precision with regard to the event log. The difference between log-precision and system-precision is extensively discussed by works [BDA14; Jan+16].
- Our method does not consider the entire behavior of the process model for the measurement of precision, i. e., we adopt the concept of *escaping behavior* first proposed in [MC12]. Therefore, the precision measurement does not consider the full behavior of the area $\ominus + \oplus$. However, this limitation is necessary when confronted with process models that allow an infinite amount of behavior.

- The precision measurement of our method highly depends on the employed state function. The state function determines what behavior needs to be observed before considering a process model as precise. Here, we used a state function based on full prefixes and the latest variable assignments. This results in a very strict notion of precision. Only when every possible path is observed with every possible variable assignment defined by the process model is observed in the event log, then, the process model considered as precise. Since our method allows to chose the state notion based on the application area, we do not consider this as a serious shortcoming.
- We first repair the event log to be fully fitting the process model before measuring precision. In extreme cases this can lead to an inherently unreliable precision measurement. For example, in the extreme situation in which the event log is not fitting at all (e. g., the event log is recorded for a different process) our method still returns a precision value. Therefore, the fitness score should be taken into account together with the precision score, e. g., by combining both measures to one score using the harmonic mean¹⁴ to penalize such extreme cases.

6.6.3 Future Work

It is not straightforward to apply the generic multi-perspective precision measure to variables defined over an infinite domain, e. g., timestamps that are used in constraints on the *time perspective* or the amount of a loan application. To apply the introduced measure to such variables, the set of possible time values would need to be discretized and made finite first. Values could be discretized and made finite by using the values that were actually observed in the event log as guidelines for the values that can be expected.

¹⁴Similar to the F-score used for classification techniques.

Part III

Multi-perspective Discovery and Enhancement

Chapter 7 We position multi-perspective process discovery and enhancement in the context of process mining.

Chapter 8 We present the Data-aware Heuristic Miner (DHM), a process discovery method that uses classification techniques to distinguish conditional infrequent paths from noise. Data- and control-flow perspective of process models are discovered together. This chapter is based on the publication [Man+17].

Chapter 9 We present the Guided Process Discovery (GPD) method, which makes use of domain knowledge about the execution of a process. Before discovering a process model, events of the event log are grouped together based on behavioral activity patterns. Activity patterns capture domain knowledge on the function perspective of a process. This chapter is based on the publications [Man+16c; Man+18].

Chapter 10 We present a decision-mining method that discovers, potentially, overlapping data rules. Overlapping rules may arise due to non-deterministic decision making or missing information in the event log. The method aims at returning process models with better fitness according to the event log at the expense of precision. This chapter is based on the publication [Man+16b].

7 Introduction to Multi-perspective Discovery and Enhancement

In Part II of this thesis, we elaborated on techniques that check the conformance between process models and event logs. Part III of this thesis focuses on *process discovery* and *enhancement* techniques. This chapter is structured as follows. In Section 7.1, we introduce process discovery and enhancement. In Section 7.2 we discuss challenges that process discovery methods face and in Section 7.3, we elaborate on our view on multi-perspective process discovery.

7.1 Process Discovery and Enhancement

Figure 7.1 illustrates the difference between process discovery and enhancement methods: Process discovery methods create process models without prior knowledge of existing models whereas enhancement methods assume an existing process model. In the next two sections we introduce both in more detail.

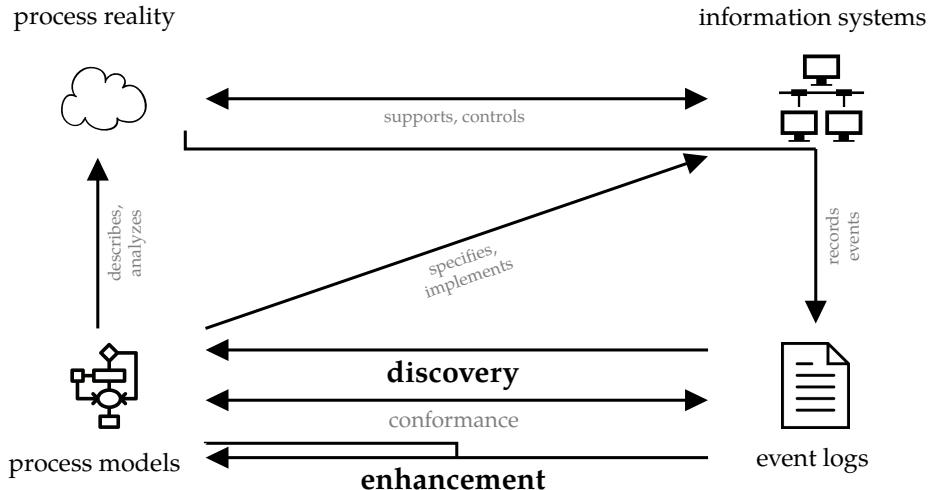


Figure 7.1: Discovery and Enhancement in the context of process mining [Aal16].

7.1.1 Process Discovery

Process discovery methods automatically generate process models that describe the real execution of a process. Process discovery methods *automatically create process models* that describe the execution of a process based on data stored in event logs. A core assumption of process discovery methods is that event logs contain events recorded for a specific process (cf., Figure 7.1). Information systems often record information about the execution of activities. This recorded execution data can be transformed into event logs [Aal16]. Each log trace of an event log contains events recorded by one execution of the process (i.e., one trace corresponds to one process instance). The goal of process discovery methods is to derive an *accurate description of the process* from the event log. Most discovery methods return process models as compact and precise description of process behavior. When analyzing the processes of an organization without a-prior knowledge, process discovery is the initial exploratory step of the analysis. Other analysis methods often rely on a model of the process, e.g., the conformance checking methods presented in Part II of this thesis. Therefore, process discovery has received a lot of attention from both academia and practice.

Among the first process discovery methods there was work by Cook et al. [CW98a; CW98b] and Agrawal et al. [AGL98]. Since then, a large number of methods that focus on the *discovery of the control-flow perspective* of a process model have been proposed:

- the α -algorithm [AWM04] and its extensions [Wen+07; Wen+10];
- approaches based on Petri net region theory (e.g., the work in [CCK08] and ILP miner [Wer+09]);
- approaches based on various heuristics (e.g., Heuristic Miner [WR11], Fuzzy Miner [Gün09]);
- methods based on genetic algorithms (e.g., Genetic Miner [Med06] and Evolutionary Tree Miner [Bui14]);
- divide-and-conquer methods (e.g., Inductive Miner [Lee17] and Constructs Competition Miner [Red+14]);
- methods based on inferring negative events that are not part of the process behavior [Goe08]; and
- several methods to discover declarative process models (e.g., Minerful [CM15], DecMiner [Che+09], and SQL-based discovery [Sch+16c]).

Since we only aim to illustrate the relevance of process discovery, we refer to [Aug+17; Wee+12] for a systematic comparison of the available process discovery technique.

Next to the academic interest, several *commercial process discovery tools* have been developed and are successfully used in practice: e.g., Fluxicon Disco¹⁵, Perceptive

¹⁵<http://www.fluxicon.com/>

Process Mining¹⁶, Celonis¹⁷, Process Mining Factory¹⁸, and Processgold Enterprise Platform¹⁹. This interest from the research community and from commercial vendors illustrates that process discovery is both a challenging research problem and highly relevant in practice.

7.1.2 Process Enhancement

Process enhancement is the category of process mining that is concerned with improving existing process models as depicted in Figure 7.1. The input to an enhancement method is an existing process model and an event log. The process model is enhanced with information that is obtained from the event log. Examples of process mining methods that can be categorized as enhancement method are the projection of performance indicators on the process models [Adr14; Roz+09], building of queuing models on top of existing process models [Sen+14], the repair of existing process models in order to better reflect the real process execution [FA15], and the extension of process models with decision logic governing the routing of cases [LA13b; RA06].

Next, we introduce some of the challenges that process discovery methods face. Note that those challenges are also faced by enhancement methods. The only difference is that there is an existing process model that should be improved. For sake of a more compact presentation, we only describe them for process discovery methods.

7.2 Challenges for Process Discovery Methods

All process discovery methods aim to return **good process models**. Therefore, for any discussion about process discovery, it is crucial to define what a good model *encompasses*. Generally, the goal is to discover a process model that rediscovered exactly the actual process under consideration. Figure 7.2 depicts the relation between the behavior specified by the discovered process model, the behavior observed in an event log recorded by the process, and the possible behavior of the actual process. To rediscover the actual process the highlighted area $\textcircled{A} + \textcircled{B} + \textcircled{E} + \textcircled{C}$ needs to be estimated. The behavior of the discovered process model ($\textcircled{A} + \textcircled{B} + \textcircled{E} + \textcircled{D}$) and the behavior of the actual process system ($\textcircled{A} + \textcircled{B} + \textcircled{E} + \textcircled{C}$) should coincide. It is possible to conduct experiments using synthetic log generated from known model to test the rediscovery performance of a process discovery method. However, in practical settings there is no gold-standard model which could be used to exactly determine the quality of the discovered model. Often, the exact process is not known and may

¹⁶<http://www.lexmark.com/>

¹⁷<http://www.celonis.com/>

¹⁸<http://www.icris.nl/>

¹⁹<http://www.processgold.com>

change over time in an uncontrolled manner. In these situations, process discovery is valuable.

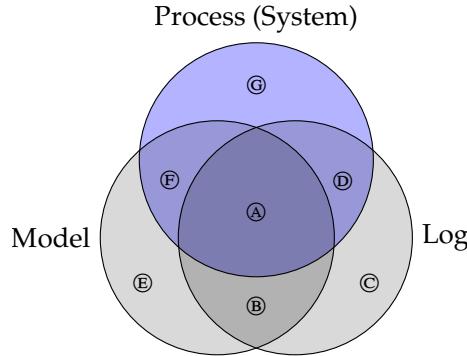


Figure 7.2: Goal of process discovery: discover the, unknown, original process system.

In Chapter 4, we described how conformance checking techniques estimate the quality of a process model based on four frequently used quality dimensions:

- *fitness*, the model should describe the observed behavior;
- *precision*, the model should describe not much more than what was observed;
- *generalization*, the model should also describe behavior that was not observed, but is likely to be part of the valid process behavior;
- *simplicity*, the model should not be unnecessarily complex.

Process discovery methods face several challenges to reach their goal: the discovery of a fitting, precise, general, and simple process model. We discuss three challenges faced by process discovery techniques that often arise in the context of real-life situations: *incompleteness*, *noise*, and *granularity*. Figure 7.3 illustrates the challenges, which are all due to the nature of the information recorded in real-life event logs. We do not claim that these three challenges are the only challenges for process discovery methods. Indeed, there are more challenges that process discovery methods face, e. g., to detect whether activities are executed in parallel, which are not in the scope of this thesis.

7.2.1 Incompleteness

Typically event logs do not contain all possible behavior of the underlying process. This can be motivated by the fact that it is unlikely that a process is executed in all possible manners. For example, a process with 10 concurrent activities can be executed in $10! = 3,628,800$ different ways. Thus, to observe all possible behavior of that process (i. e., $\textcircled{G} = \textcircled{F} = \emptyset$), the event log would need to contain all those 3.6 million distinct traces. Even in an event log with 3.6 millions traces, it is extremely

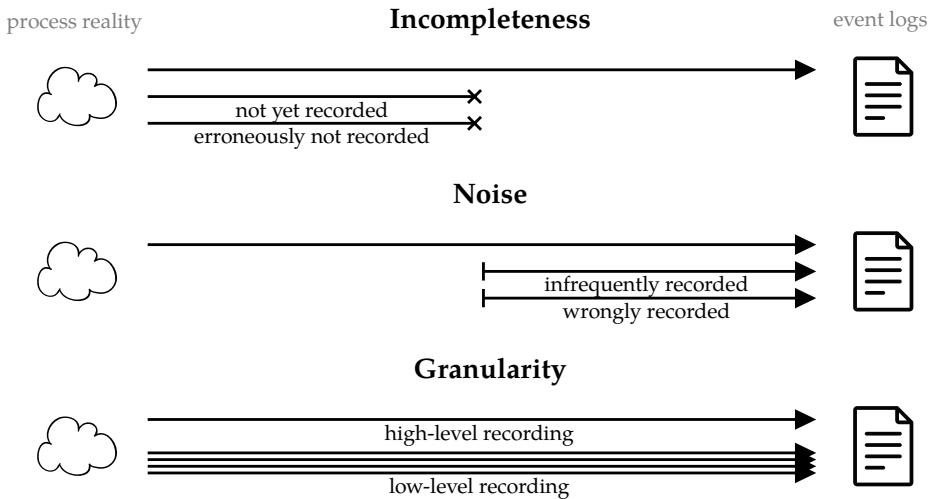


Figure 7.3: Incompleteness, noise, and the granularity of events are challenges for process discovery methods when working with real-life event logs.

unlikely that all those distinct traces are present [Aal16]. Some of the execution variants will be observed more than once, other variants will never be observed. When adding other process perspectives and loops this *incompleteness* problem grows further. In case of loops the number of potential execution traces of a process is infinite. Therefore, process discovery algorithms need to generalize from the observed behavior.

7.2.2 Noise

Another important challenge for process discovery methods is to handle event logs that contain *noise* [Aal16; Wee+12]. Some definitions of noise only entail outliers that were recorded due to errors [AGL98], others also include low-frequent behavior. Examples for events that are typically considered noise are:

- events recorded by activities that were executed out of the normal order,
- events recorded due to rare exceptions,
- events recorded due to temporary workarounds,
- events wrongly recorded or lost due to logging errors, system malfunction or other data quality issues.

However, in a real-life setting, without a-priori knowledge on the process, it is difficult to distinguish between data quality problem (e. g., recording errors, lost data) and valid events that are recorded infrequently. We consider *noise* to be any event that corresponds to *undesirable infrequent behavior* [Aal+12; Aal16; BMA13;

CLH16; Sur+17]. What is considered undesirable behavior depends on the application setting. When looking at the mainstream behavior of the process, then, all infrequent behavior is undesirable. When looking for workaround and divergent process executions, some infrequent behavior may be desirable.

7.2.3 Granularity

Most process discovery methods assume that recorded events correspond to meaningful activities in the instances of a process. However, events may be recorded on different levels of granularity [Bai15]. Some events may refer to activities on a high level of abstraction. Their execution is easily recognizable for process workers (e. g., a nurse reacts to an alarm raised by a patient). Other events may be recorded on a lower level of abstraction. Multiple of such low-level events may correspond to a recognizable high-level activity. For example, the high-level activity *Patient Alarm* may be recorded as a series of low-level events: the patient presses the alarm button, the nurse enters the room, the nurse clears the alarm state, and the nurse leaves the room. When discovering processes based on those low-level events, the resulting process model may be unrecognizable for process workers. The discovered model gives a description at the wrong level of abstraction. Therefore, the discovered model is useless, similarly to using a street map when trying to fly a place.

7.3 Multi-perspective Process Discovery

So far, we have only considered the discovery of the control-flow perspective of the process. Indeed, the control-flow can be seen as the backbone of a process. A good understanding how activities relate to each other in terms of ordering and dependencies is an important prerequisite for many analysis tasks.

However, as motivated multiple times previously in this thesis (cf., Sections 1.2 and 4.2), the control-flow perspective is only one of the potential viewpoints on a process. In this section, we extend our view towards multiple-perspectives and give an overview of the state-of-the-art methods. Multi-perspective process discovery methods consider information on the data, resource, and time, and function perspective of the process.

One challenge when considering multiple perspectives on processes lies in the provision of data that can be linked to the process execution. Often, data is fragmented across various sources and needs to be consolidated [Aa+15], e. g., by integrating events from different source information systems [MLR15a]. With emerging trends such as the Internet of Things (IoT), i. e., the connection of all physical objects to the network, more and more data on the execution of a process will be available. Therefore, information extracted from this recorded data can be connected to the process execution and, thus, used for process discovery [Aal15].

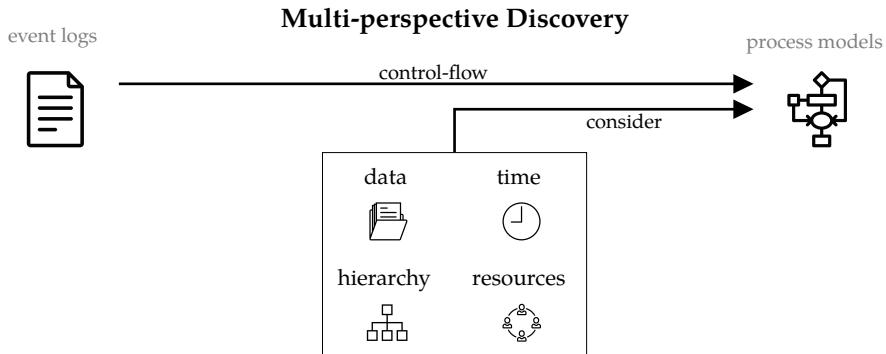


Figure 7.4: Multi-perspective process discovery methods use information about the context of process to obtain a more complete picture of the process.

Several methods that discover process perspectives different from the control-flow have been proposed. Some examples are methods that:

- discover social networks [ARS05],
- discover decision rules and decision models [BBW16; LA13b; LDG13; RA06],
- discover queues [Sen+15], and
- discover GANTT charts [Bal+15].

However, those methods follow a staged approach. Only one process perspective is discovered (e.g., the social network describing the resource perspective). In this thesis, we aim to develop methods in which multiple perspectives are intertwined.

In Chapter 8 and Chapter 9, we present two multi-perspective discovery methods that discover *integrated models* in which multiple perspectives on the process are **intertwined with the control-flow**. The goal is to consider multiple perspectives on the process for the discovery of a single model. The control-flow perspective is influenced by the data, resource, time, and functional perspective. This is different from the staged approach that the previously mentioned methods follow, in which perspectives are discovered step-by-step in isolation.

Figure 7.4 illustrates the goal to discover the different perspectives in an integrated manner. The semantics of events may depend on context information on other perspectives. Depending on the context the same event may relate to different activities, the importance of events may differ, etc. Thus, process discovery methods that ignore the context in which an event occurred, fail to reveal those differences. The complete picture of the process is not discovered and, often, imprecise process models are returned. There are many possible definitions of what the context of a process entails. The context may consist of, e.g., the sequence of activities that occurred beforehand, the data values that were recorded beforehand, the resources available at the time of execution, seasonal effects, and many more possible influ-

ence factors.

There are also methods that combine results discovered from several perspectives into integrated models, i. e., follow the integrated approach taken in this thesis. Here, we sketch briefly some examples of those of multi-perspective discovered methods.

- Rozinat [Roz10] describes an approach in which the discovered control-flow is enriched with decision rules, performance information (i. e., waiting time and execution time distributions), and organizational information (i. e., roles and originators). The result is a Colored Petri Net that integrates three discovered perspectives: the control-flow perspective, the data perspective, and the resource perspective. In this case the discovered information is discovered separately and only afterwards integrated.
- Maggi et al. present a method to directly discover multi-perspective declarative process models, i. e., sets of rules that cover the control-flow and data perspective are discovered [Mag+13; Sch+16b];
- Schöning et al. present a method to discover rules expressed in the declarative language DPIL that concern both the resource and control-flow perspective in [Sch+16a].
- Van Eck et al. describe a method in which multiple perspectives on one process based on the notion of process states are discovered, control-flow and state information is intertwined [ESA16];
- Conforti et al. present the BPMN Miner, a tool which discovers functional dependencies between activities and the control-flow of the process together. This proposal integrates the control-flow and the function perspective in an integrated process model [Con+16].

We discuss the research that is more closely related to our proposed methods later in each chapter systematically and in more detail.

In this thesis, we propose two multi-perspective discovery methods and one multi-perspective enhancement method.

- Chapter 8 presents a method that uses data recorded on multiple process perspectives to address the *noise challenge*. The method aims to distinguish infrequent paths from random noise by using classification techniques. Data- and control-flow are learned together.
- Chapter 9 presents a method that uses domain knowledge encoded in multi-perspective activity patterns to address the *granularity challenge* and can handle *noise*. The method lifts low-level events to high-level activities and discovers hierarchical multi-perspective process models.
- In Chapter 10 we introduce process enhancement and present a method to enhance an existing process model with decision logic that constraints the routing of process instances.

8 Data-aware Heuristic Process Discovery

Using all the recorded behavior in the event log (i. e., including noise) for process discovery often leads to complex models that are unusable for the purpose of process analysis [Aal16; Sur+17]. Therefore, *noise filtering methods* that distinguish noise from the desirable, regular behavior of the process have been developed. These aim to deal with the complex behavior recorded in real-life event logs. What is considered as *desirable behavior* depends on the context of the process mining analysis. We categorize two different types of analysis:

- analysis of frequent behavior (e. g., process highways) and
- analysis of infrequent behavior (e. g., workarounds and deviations).

Often the focus of process discovery is on *frequent behavior*, i. e., the goal is to discover *process highways* that are able to explain the dominant behavior. In this context, all infrequent events are undesirable since they might negatively affect the quality of the model. Including many infrequent process paths in the discovered model might lead to overly complicated models. However, when the focus of process discovery is on analyzing *workarounds, deviations, and other infrequent behavior*, some infrequent paths and events are part of the behavior that needs to be discovered. Otherwise, the process model would be unsuitable to analyze this kind of behavior.

This chapter presents a data-aware heuristic process discovery technique that aims to discover an end-to-end process model that includes possibly interesting infrequent behavior while filtering random noise that would not yield any insight into the process. We leverage the data perspective to identify behavior that can be characterized by deterministic rules (i. e., conditions) over the data attributes recorded by the process. Many processes are governed by rules. Decisions are taken on the basis of available data, available resources, and the process context. The idea is that some infrequent paths may be executed rarely because the corresponding conditions are rarely fulfilled. These paths and the conditions are likely to be of great interest to process analysts (e. g., in the context of risks and fraud). For example, shortcuts in a process might only be taken by a specific resource or undesired behavior might be subject to conditions. This kind of *infrequent conditional process behavior* should not be set aside as *noise*.

The remainder of this chapter is structured as follows. In Section 8.1, we present a motivating example. In Section 8.2, we give an overview of our method. In Section 8.3, the core method, i. e., the discovery of conditional infrequent process

behavior, is presented. Section 8.4 describes an extension with to multi-perspective process models. In Section 8.5 we evaluate the noise filtering capabilities of our method with synthetic event logs and in Section 8.6 we compare it with related work. Finally, in Section 8.7 we conclude the chapter.

8.1 Motivation for Discovering Conditional Behavior

We start with a motivational example that is used throughout this chapter. Assume the hospital process that is used as running example throughout this thesis. Based on the textual description of the process in Example 1.1, there are several cases of interesting infrequent behavior:

- (A) Only in **exceptional cases**, patients are assigned the triage color *white*. Those patients typically, *leave* the emergency ward after being registered.
- (B) There are two different work practices regarding the activities Diagnostics and Visit.
 - a) Normally, the doctor first visits the patients and, *thereafter*, the diagnostic test is conducted.
 - b) **Sometimes**, both activities are executed in *reversed order*: first a medical diagnostic is taken and, only after that, the doctor visits the patient.
- (C) For a **specific group** of patients, those that are transferred to another hospital, an ambulance needs to be organized.

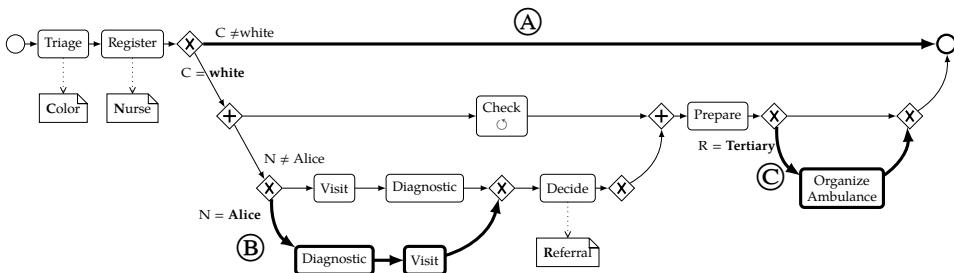
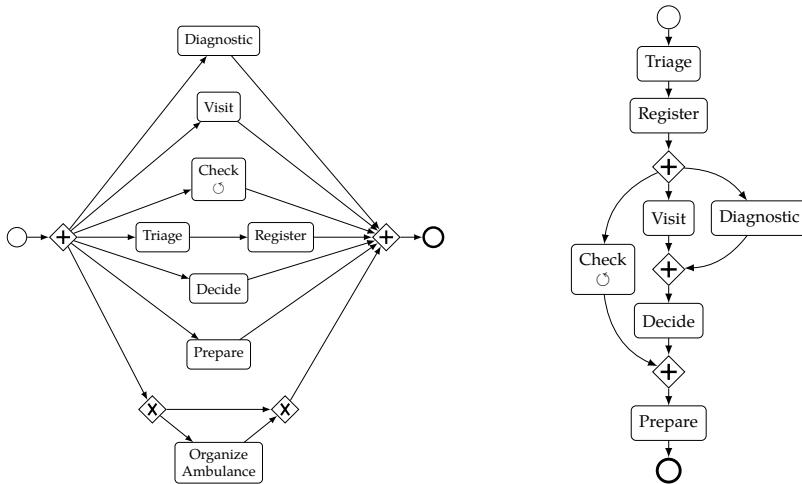


Figure 8.1: Infrequent process behavior highlighted on a BPMN model of the hospital process.

The BPMN model in Figure 8.1 depicts a variant of the running example hospital process (cf., Example 1.1) that we use to explain the effect of infrequent behavior on process discovery and, subsequently, our new discovery technique. We removed some behavior of the original process model (cf., Figure 3.4) since we want to simplify the presentation of our technique.²⁰ This process contains three examples of

²⁰We removed the exclusive choice between activities *Transfer*, *Discharge* and *Observe* at the end of the



(a) The Inductive Miner filters little of the injected noise and fails to reveal behavior ④ and ⑧.

(b) The Heuristics Miner filters the injected noise well, but fails to show behavior ④, ⑧ and ⑨.

Figure 8.2: Models discovered by Inductive Miner and Heuristics Miner on an log generated from the hospital process with randomly injected out-of-order events in 5% of the traces.

infrequent, data-dependent behavior: a data-dependent path (④), data-dependent re-sequencing (⑧), and a data-dependent activity (⑨). The goal of our work is to rediscover such behavior, while ignoring random noise.

Assume that an event log recording the process behavior of the BPMN model shown in Figure 8.1 was obtained from information systems of the hospital. The three examples for infrequent behavior are recorded in only 1.4% (④), 33.4% (⑧), and 1.7% (⑨) of the process instances. As motivated in the introduction, it is likely that this event log contains noise. We generated a perfectly fitting event log based on the process model shown in Figure 8.1 and introduced a controlled degree of noise. In 5% of the cases one additional event was introduced at a random position in the log trace.

We applied both the Heuristics Miner²¹ [WR11] and the Inductive Miner²² [LFA13] as representatives of process discovery methods that support noise filtering. Figure 8.2 shows the resulting process models of both discovery methods. We converted both process models into BPMN models to facilitate the comparison. Both methods are unaware of the data perspective. Moreover, they fail to distinguish between the injected random noise and the infrequent data-dependent behavior

process and the loop structure regarding activities *Diagnostic* and *Visit*

²¹The Heuristics Miner with an observation threshold of 0.1 was used.

²²The Inductive Miner infrequent variant with the default noise filtering setting of 0.2 was used.

Ⓐ, Ⓑ, and Ⓒ. It might be possible to tweak the parameters of the algorithms to reveal more of the infrequent behavior (e.g., through grid search). However, finding the correct parameter setting that does not include unrelated noise requires deep knowledge about the underlying process. This is often infeasible.

Moreover, based on the process models in Figure 8.2 it is impossible to reveal the infrequent data-dependent behavior by using decision mining techniques. Those techniques can only reveal decision rules for paths that are reflected in the process model, i.e., exclusive split gateways that model a decision between the process paths need to be included in the model. Thus, low-frequent but deterministic behavior remains undetected. The method proposed in this chapter, discovers all three types of infrequent conditional process behavior depicted in Figure 8.1 and successfully filters the random noise added. Next, we are going to describe our technique.

8.2 Overview of the Data-aware Heuristic Process Discovery Method

We present a method for **data-aware heuristic process discovery**. The method extends the ideas of the Heuristics Miner [WR11] with the use of classification techniques to discover the control-flow and the data perspective of the process together. It reveals data dependencies between activities, and uses these data dependencies to *distinguish noise from infrequent conditional behavior*, i.e., reveals behavior that would otherwise have been dismissed as noise by the Heuristics Miner.

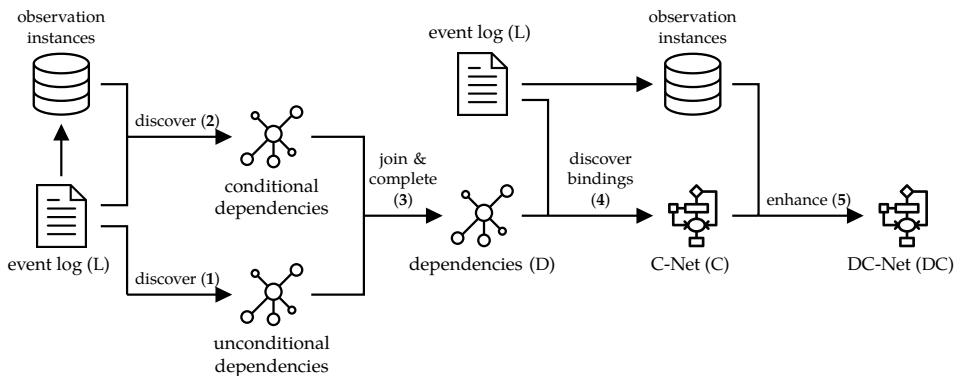


Figure 8.3: Overview of the proposed data-aware heuristic process discovery method.

Figure 8.3 gives an overview of the method. The required input is an event log and several parameters that are used to configure the thresholds of the employed heuristics. Moreover, we assume that the event log contains attributes encoding information about the *context* of the process, i.e., resources executing activities,

data attributes, and time information. Based on this input, the method consists of the following five steps:

1. We discover a set of unconditional dependencies based on the heuristics proposed by the Heuristics Miner.
2. We train classifiers to obtain an additional set of conditional dependencies, which may be infrequent and, thus, have been dismissed by the Heuristics Miner.
3. We join both sets of conditional and unconditional dependencies and add dependencies for disconnected activities.
4. We discover the bindings (i.e., the split and join behavior) and create a Causal Net (C-Net) (cf., Definition 3.13) based on the approach proposed by the Heuristics Miner [WR11].
5. We enhance the C-Net to a multi-perspective Data Causal Net (DC-Net) adding discovered decision rules.

In Section 8.3, we describe steps 1-4 and in Section 8.4 step 5 of the proposed method.

8.3 Discovering Conditional Behavior

We describe how to discover conditional behavior using classification techniques and present an integrated discovery method that returns C-Nets.

8.3.1 Dependency Conditions and Dependency Measure

To discover data-dependent behavior in the event log, we make use of classification techniques (e.g., decision trees such as C4.5 [Qui93]). More specifically, we rely on binary classifiers that predict the occurrence of directly-follows relations based on attribute values recorded in the event log. We denote these binary classifiers as *dependency conditions*: the classifier encodes the condition under which a dependency between two activities was observed in the event log.

Definition 8.1 (Dependency conditions). Let U be the universe of values. Let V_L be a set of attributes and let $\text{dom}(v) \in \mathbb{P}(U)$ be the domain of attribute $v \in V_L$. Let $\Sigma \subseteq U$ be a set of activities. We define a dependency condition function:

$$dc : (\Sigma \times \Sigma) \rightarrow (\mathcal{U}_{\text{dom}}^L \rightarrow \{0, 1\})$$

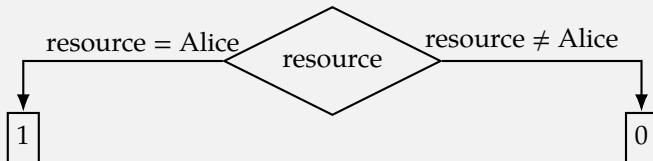
that returns the dependency condition for any two activities.²³ A *dependency condition* $dc(a, b)$ is a binary classifier that predicts whether an event recording the execution of activity a is followed directly by an event recording the execution of

²³Note that $\mathcal{U}_{\text{dom}}^L$ denotes a partial attribute assignment function as introduced in Definitions 2.7 and 2.10, i.e., some attributes in $v \in V_L$ are assigned values from their domain $\text{dom}(v) \subseteq U$.

activity **b** based on the attribute assignment $w \in \mathcal{U}_{dom}^L$, i.e., $dc(a, b)(w) = 1$ when **b** is predicted to directly follow **a** and $dc(a, b)(w) = 0$ when a *different* activity is predicted. \diamond

We introduce the shorthand notation $dc_{a,b} = dc(a, b)$. Moreover, we denote with $1_{a,b}$ a special dependency condition function that returns classifiers predicting 1 regardless of the attribute values, i.e., $\forall_{a,b \in \Sigma} \forall_{w \in \mathcal{U}_{dom}^L} (1_{a,b}(w) = 1)$.

Example 8.1 (Dependency conditions). In the remainder of this chapter, we use an example event log L_h that consists of 150 traces: 50 traces record the same sequence of activities and attributes as in σ_{h1} , 50 traces are like σ_{h2} and 50 traces are like σ_{h3} (see Table 8.1). Based on event log L_h , the dependency condition function dc could return the following decision tree (i.e., binary classifier) as dependency condition $dc_{D,V}$ for activities *Diagnostics* (D) and *Visit* (V):



The dependency condition predicts, based on the attributes values recorded up to and including to the occurrence of *Diagnostic*, whether the next activity is likely to be *Visit* (i.e., predicts class 1) or any other different activity (i.e., predicts class 0). Here, the dependency condition makes its prediction by virtue of the value recorded for the attribute $resource \in V_L$, which stores the name of the nurse that executed activity *Register* in the example log L_h . If *Alice* registered the patient, the next activity is predicted to be *Visit*; otherwise, a different activity follows. We employ decision trees; however, any binary classifier would be suitable. We show later in Section 8.3.2 how to build a classifier based on the event log.

Given a dependency condition, we establish the frequency with which activities **b** are observed to directly follow an activity **a** in the event log when the condition holds for the attributes values observed before **b** occurred. Thus, the condition could have been used to predict the occurrence of activity **b** given the previously observed attribute values. We denote this as: *conditional directly follows*.

Definition 8.2 (Conditional directly follows relation). Given activities $a, b \in \Sigma$ and dependency conditions dc , we write $a >^{dc_{a,b}, L} b$ if and only if an execution of activity **a** is *directly followed* by an execution of activity **b** that is recorded as event e under dependency condition $dc_{a,b}(\text{latest}(e))$, i.e., the latest attribute assignment before activity **b** was observed (cf., Definition 2.11) can be used to predict the

Table 8.1: Three exemplary traces of an event log L_h recorded by the hospital process.(a) Trace σ_{h1}

id	activity	color	resource	referral
e_{h10}	Triage	Red		
e_{h11}	Register		Joe	
e_{h12}	Check			
e_{h13}	Check			
e_{h14}	Check			
e_{h15}	Visit			
e_{h16}	Diagnostic			
e_{h17}	Decide			Ward
e_{h18}	Prepare			

(b) Trace σ_{h2}

id	activity	color	resource	referral
e_{h20}	Triage	Red		
e_{h21}	Register		Alice	
e_{h22}	Check			
e_{h23}	Diagnostic			
e_{h24}	Visit			
e_{h25}	Check			
e_{h26}	Decide			Tertiary
e_{h27}	Prepare			
e_{h28}	Organize Ambulance			

(c) Trace σ_{h3}

id	activity	color	resource	referral
e_{h30}	Triage	Red		
e_{h31}	Register		Joe	
e_{h32}	Check			
e_{h33}	Visit			
e_{h34}	Diagnostic			
e_{h35}	Check			
e_{h36}	Check			
e_{h37}	Decide			Ward
e_{h38}	Prepare			

occurrence of **b**. We denote the frequency of a conditional directly-follows relation $a >^{dc_{a,b}, L} b$ in the event log as:

$$|a >^{dc_{a,b}, L} b| = |\{e \in E \mid \exists e' \in E \ (\#_{act}(e') = a \wedge e' = \bullet e \wedge \#_{act}(e) = b \\ \wedge dc_{a,b}(\text{latest}(e)) = 1)\}| \quad \diamond$$

We use the conditional directly-follows relation to define a data-aware variant of the dependency measure that was originally proposed by the Heuristics Miner in [WR11].

Definition 8.3 (Conditional dependency measure). Given activities $a, b \in \Sigma$ and dependency conditions dc . We define $a \Rightarrow^{dc, L} b \in [-1, 1]$ as the strength of the causal dependency from **a** to **b** under condition $dc_{a,b}$ in the event log:

$$a \Rightarrow^{dc, L} b = \begin{cases} \frac{|a >^{dc_{a,b}, L} b| - |b >^{dc_{a,b}, L} a|}{|a >^{dc_{a,b}, L} b| + |b >^{dc_{a,b}, L} a| + 1} & \text{for } a \neq b, \\ \frac{|a >^{dc_{a,b}, L} a|}{|a >^{dc_{a,b}, L} a| + 1} & \text{otherwise.} \end{cases} \quad \diamond$$

The intuition behind the data-aware variant of these heuristic measures is that a relation (a, b) should be included in the discovered process model when it is clearly characterized by a dependency condition $dc_{a,b}$.

Example 8.2 (Conditional dependency measure). Assume to be given the example event log L_h introduced in Example 8.1. We want to determine the conditional dependency measure $D \Rightarrow^{dc, L} V$ from activity Diagnostic (D) to activity Visit (V). Assume that we obtained a dependency condition $dc_{D,V}$ that returns 1 only if attribute resource values takes on the value Alice. Then, we obtain the number of times that activity Diagnostic is directly followed by activity Visit under condition $dc_{D,V}$ as $|D >^{dc_{D,V}, L} V| = 50$, and the number of times activity Visit is directly followed by activity Diagnostic under condition $dc_{D,V}$ as $|V >^{dc_{D,V}, L} D| = 0$.

Therefore, the conditional dependency measure under conditions dc is $D \Rightarrow^{dc, L} V = \frac{50-0}{50+0+1} \approx 0.98$. This dependency measure indicates a strong dependency relation from activity Diagnostic to activity Visit under the condition $dc_{D,V}$. By contrast, if we consider the unconditional dependency measure, then we obtain $D \Rightarrow^{1,L} V = \frac{50-100}{50+100+1} \approx -0.33$. Thus, when disregarding the data perspective, both activities appear to be executed in parallel.

8.3.2 Discovering Dependency Conditions

We described the *conditional directly-follows relation* and the *conditional dependency measure*. The latter measure is used to determine which relations should be included

in the C-net that we want to discover. Both concepts rely on *dependency conditions* that need to be discovered from the event log.

We train a binary classifier that can be used as dependency condition for source activity $a \in \Sigma$ and target activity $b \in \Sigma$. Binary classifiers require the definition of a positive and a negative class. In our case, executions of activity b are considered as positive and executions of other activities that are not in parallel to a are considered as negative.

Definition 8.4 (Negative candidate activities). Given a source activity $a \in \Sigma$, a candidate activity $b \in \Sigma$, and a dependency threshold $\theta_{dep} \in [0, 1]$, we denote with $\Sigma_a^{NEG}(\theta_{dep})$ the set of negative candidate activities x that directly follow a in the event log for which the unconditional dependency measure is above the threshold: θ_{dep} :

$$\Sigma_a^{NEG}(\theta_{dep}) = \{x \in \Sigma \mid (a \Rightarrow^{1,L} x) \geq \theta_{dep}\} \setminus \{b\}. \quad \diamond$$

To train a classifier, we need to define a set of training or observation instances. We build a set of observation instances for every combination of activities $(a, b) \in \Sigma \times \Sigma$ of activities that are recorded in the event log. Events recording the execution of activity b are added as instances for the positive class and events recording the execution of activities in $\Sigma_a(\theta_{dep})$ are added as instances of the negative class.

Definition 8.5 (Observation Instance Function). Let $C = \{1, 0\}$ be the set of target classes. Let $L = (E, \Sigma, \#, \mathcal{E})$ be an event log. Let $E_L(a, b, \theta_{dep}) \subseteq E$ be the subset of events that directly follow an event that records an execution of activity $a \in \Sigma$ and refer to negative candidate activities in $\Sigma_a(\theta_{dep})$ or to the positive candidate activity $b \in \Sigma$, i. e.,

$$E_L(a, b, \theta_{dep}) = \{e \in E \mid \#\text{act}(\bullet e) = a \wedge \#\text{act}(e) \in (\Sigma_a^{NEG}(\theta_{dep}) \cup \{b\})\}.$$

We define function $OI_L : (\Sigma \times \Sigma \times [0, 1]) \rightarrow \mathbb{B}(\mathcal{U}_{dom}^L \times C)$ to return multi-sets of observation instances²⁴ for any pair of activities:

$$OI_L(a, b, \theta_{dep}) = \biguplus_{e \in E_L(a, b, \theta_{dep})} [(latest(e), class(e))] \\ \text{where } class(e) = \begin{cases} 1, & \text{if } \#\text{act}(e) = b \\ 0, & \text{otherwise.} \end{cases} \quad \diamond$$

Conceptually, our method is independent of the used classification algorithm. Concretely, we employ decision trees (C4.5) [Qui93] as an efficient method that result in human interpretable conditions. We build the dependency conditions dc by assembling a set of observation instances $OI_L(a, b, \theta_{dep})$ and, then, use a C4.5 decision tree builder $buildTree_{C, mi}(OI) \in \mathbb{P}(\text{EXPR}(V_L) \times C)$ to build a decision tree for each possible relation $(a, b) \in \Sigma \times \Sigma$.

²⁴The concept of observation instances is introduce in Definition 2.12.

Example 8.3 (Collecting observation instances). Assume the dependency threshold $\theta_{dep} = 0.9$ and the same event log L_h as in Example 8.2. We train a classifier for the dependency condition $dc_{D,V}$, i. e., the dependency relation from Diagnostic (D) to Visit (V) using the observation instances $OI_L(D, V, \theta_{dep})$. First, we identify the negative candidate activities $\Sigma_D^{NEG}(\theta_{dep})$, i. e., those activities that are different from activity Visit and that are observed to occur not in parallel to activity Diagnostic. Set $\Sigma_D^{NEG}(\theta_{dep})$ is determined based on the user-specified dependency threshold $\theta_{dep} = 0.9$. In this case, we obtain $\Sigma_D^{NEG}(\theta_{dep}) = \{\text{Decide}\}$. Activity Check (C) is not a negative candidate activity since the unconditional dependency measure $D \Rightarrow^{1,L} C$ is below the threshold of 0.9. The collected observation instances are $OI_L(D, V, \theta_{dep}) = [((\text{color} \rightarrow \text{Red}, \text{resource} \rightarrow \text{Joe}), \text{Decide})^{50}, ((\text{color} \rightarrow \text{Red}, \text{resource} \rightarrow \text{Alice}), \text{Visit})^{50}]$. Since activity Check is not a negative candidate activity, the instances based on trace σ_{h_3} are not included.

We train a C4.5 decision tree and obtain the dependency condition $dc_{D,V}$ with $dc_{D,V}(\text{resource} \rightarrow \text{Alice}, \dots) = 1$ and $dc_{D,V}(\text{resource} \rightarrow \text{Joe}, \dots) = 0$, i. e., the decision tree classifier described in Example 8.1 predicts that activity Visit follows directly after activity Diagnostic when the patient was registered by nurse Alice. All other attribute values are irrelevant for the decision.

Not all dependency conditions are of the same quality. We aim to distinguish those dependency conditions with good discriminative power from those with a high error rate. For this purpose, we introduce a quality score for dependency conditions.

Definition 8.6 (Quality of a Dependency Condition). Let $L = (E, \Sigma, \#, \mathcal{E})$ be an event log. Let $dc_{a,b}$ be a dependency condition for activities $a, b \in \Sigma$. We denote with

$$\text{quality}_L(dc_{a,b}) \in [0, 1]$$

the quality score of dependency condition $dc_{a,b}$ given the observations recorded in the event log. \diamond

In Definition 8.6, we abstract from the exact manner in which the quality score is determined based on the event log. We recommend to use standard evaluation techniques that avoid overfitting such as using cross validation or separating the source data into a training set (used as observation instance according to Definition 8.5) and a validation set that is used to determine the quality. Doing so reduces the risk of discovering a dependency condition that is overfitting the data, and, thus, finding many irrelevant and non-generalizing (cf., the discussion on quality measures in Section 7.2).

There are many possible existing performance measures for binary classification algorithms that can be used to estimate the quality score of a dependency condition, e. g., Accuracy, RMSE, F1-score, AUCROC [Bra97], Cohen's kappa [Coh60] and

AUK [KBP12], Matthews correlation coefficient [Pow11], etc. None of the measures is universally accepted to be used in all situations, the correct choice depends on the concrete application area [Pow11].

We opted for Cohen's kappa (κ) [Coh60] to evaluate the quality of discovered dependency conditions. It has been recommended to evaluate nonparametric binary classifiers, such as C4.5, on data with imbalanced class priors [Ben08]. Kappa was introduced to measure the agreement between two raters, which is the same situation as to measure the agreement between class predictions made by a binary classifier and the observed classes:

$$\kappa = \frac{p_o - p_c}{1 - p_c}.$$

Term p_o is the accuracy of the prediction, i. e., the agreement between the prediction and the observation. Kappa also includes term p_c , which estimates the agreement by chance that can be expected when randomly guessing. The term p_c is estimated based on the class distribution. A κ -value greater than 0 indicates that the prediction was better than by weighted random selection [Coh60]. Kappa favors a good prediction of the minority class in situations with imbalanced class distributions (i. e., as likely to be encountered in our setting) since mostly predicting the majority class would not yield better agreement than to be expected by chance. However, we do *not claim* κ to be the best measure in all cases and, thus, foresee other measures to be plugged-in to our method depending on the application area and the employed classification technique.

8.3.3 Discovering Causal Nets with the DHM

We describe how we realize our data-aware heuristic process discovery method as the Data-aware Heuristic Miner (DHM) that uses conditional dependencies and the conditional dependency measure to discovery process models in the C-net notation (cf., Section 3.3). Figure 8.4 shows the C-net that is discovered by the DHM based on an event log recorded by the hospital process as depicted in Figure 8.1.

The DHM supports four user-specified thresholds that can be used to tune the noise filtering capabilities to specific needs of the user and the situation:

- $\theta_{obs} \in [0, 1]$, is the observation threshold, which controls the required frequency of relations relative to the number of traces in the event log to be included in the model;
- $\theta_{dep} \in [0, 1]$, is the dependency threshold, which controls the required strength of the unconditional dependency measure of relations to be included in the model;
- $\theta_{con} \in [0, 1]$, is the condition threshold, which controls the required quality of the conditional dependencies to be considered for inclusion, i. e., the κ -value that is required; and

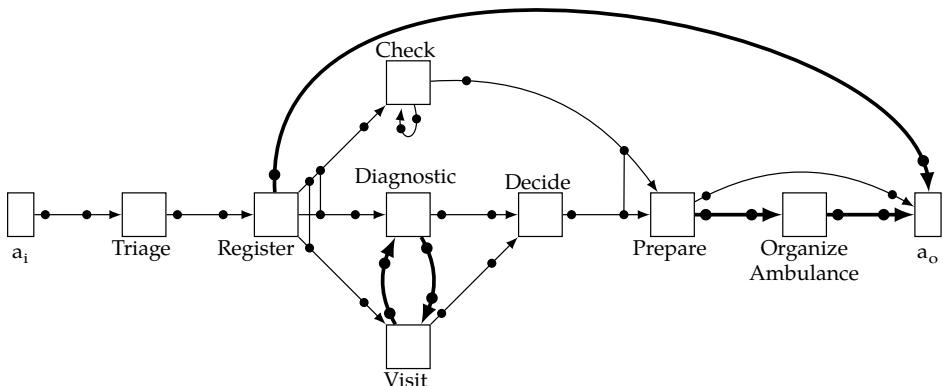


Figure 8.4: The simplified variant of the hospital process modeled as C-Net.

- $\theta_{bin} \in [0, 1]$, is the binding threshold, which controls the required relative frequency of observations of a bindings to be included in the model.

The DHM discovers a C-net $(\Sigma, a_i, a_o, D, I, O)$ from event log $L = (E, \Sigma, \#, \mathcal{E})$ and thresholds $\theta_{obs}, \theta_{dep}, \theta_{bin}, \theta_{con}$ in the following three steps:

1. we guarantee a unique start activity a_i and a unique end activity a_o ,
2. we discover the set of unconditional- and conditional dependency relations, which together form the set of dependency relations D of the C-net,
3. we determine the input bindings I and the output bindings O .

Unique Start- and End activities

We pre-process all traces of the event log by adding artificial start- and end events. This ensures that the resulting C-net has unique **start** and **end** activities. Thus, we assume that activities a_i and a_o exist, i. e., $\{a_i, a_o\} \subseteq \Sigma$ and that any trace starts with a_i , ends with a_o , and a_i and a_o do not appear at any other position, i. e., $\forall_{\sigma \in \mathcal{E}} \forall_{1 \leq i \leq n} (\sigma = (e_i, e_1, \dots, e_n, e_o) \wedge \#_{act}(e_i) = a_i \wedge \#_{act}(e_o) = a_o \wedge \#_{act}(e_i) \neq a_i \wedge \#_{act}(e_i) \neq a_o)$.

Example 8.4 (Ensure artificial start- and end activities). Take trace $\sigma_{h1} = \langle e_{h10}, e_{h11}, e_{h12}, e_{h13}, e_{h14}, e_{h15}, e_{h16}, e_{h17}, e_{h18} \rangle$ that is shown in Table 8.1. We extend trace σ_{h1} to trace $\sigma'_{h1} = \langle e_i, e_{h10}, e_{h11}, e_{h12}, e_{h13}, e_{h14}, e_{h15}, e_{h16}, e_{h17}, e_{h18}, e_o \rangle$ with $\#_{act}(e_i) = a_i$ and $\#_{act}(e_o) = a_o$. No further attributes are assigned by artificial start event e_i and artificial end event e_o .

Discovering Dependency Relations

The main challenge when discovering a C-net from an event log is to determine the set of dependency relations D . A dependency relation $(a, b) \in D$ represents a causal dependency from activity a to activity b , i.e., the execution of activity a depends on a prior execution of activity b . We adopt the main idea of the Heuristics Miner presented in [WR11] and use the dependency measure to determine the strength of a dependency relations based on how often activity a is observed to be directly followed by activity b in the event log. Strong dependency relations are likely to indicate causal dependency between activities that should be reflected in the C-net.

Differently from the Heuristics Miner algorithm, we also consider the conditional dependency measure defined in Section 8.3.1 and discover the set of dependency relations in the following three steps:

1. We build the set of unconditional **dependency relations** as follows:

$$D_{\text{udr}} = \{(a, b) \in \Sigma \times \Sigma \mid a \Rightarrow^{1,L} b \geq \theta_{\text{dep}} \wedge \frac{|a >^{1,L} b|}{|\mathcal{E}|} \geq \theta_{\text{obs}}\}.$$

2. We **discover the dependency condition function** dc by training a classifier that uses the observation instances $OI_L(a, b, \theta_{\text{dep}})$ for each pair of activities $(a, b) \in \Sigma \times \Sigma$. We use dc to discover the **conditional dependency relations** to D . As threshold, we use θ_{con} instead of θ_{obs} to also obtain infrequent, high-quality data conditions:

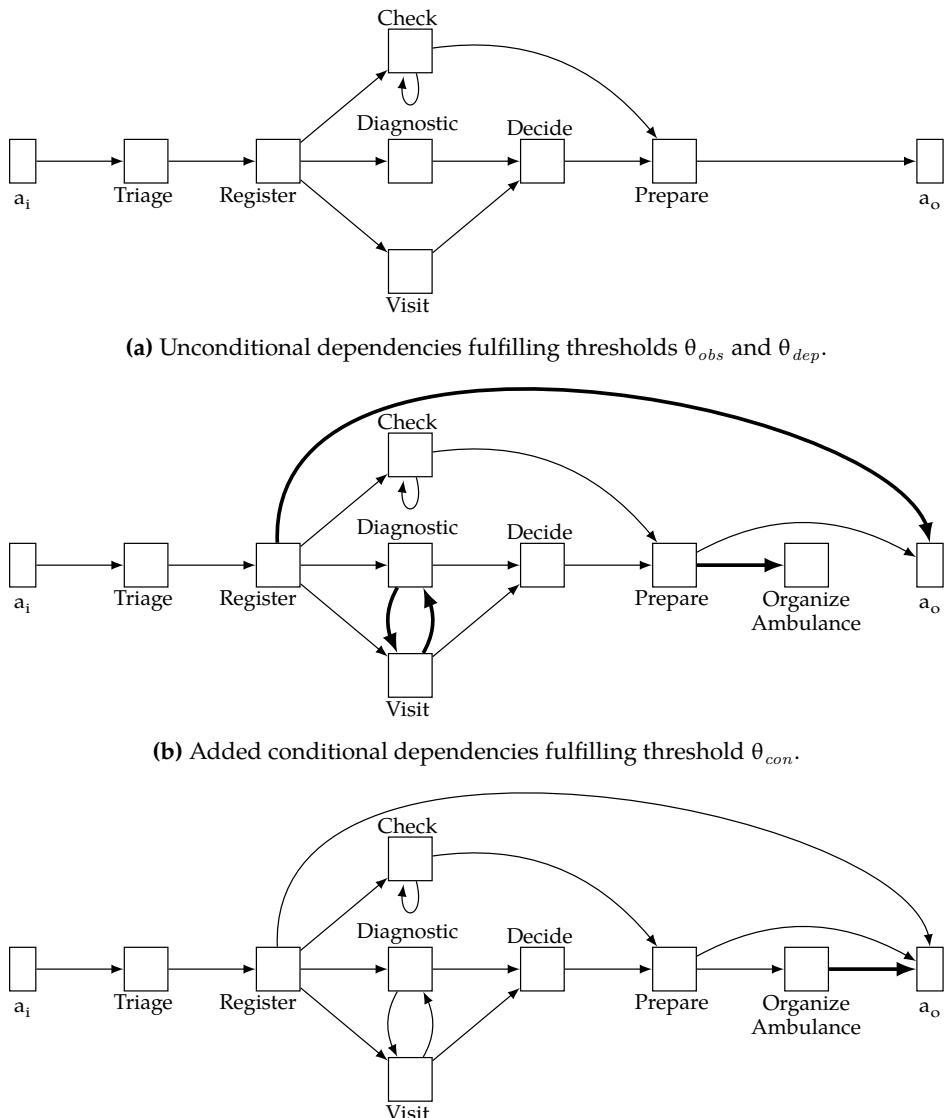
$$D_{\text{cdr}} = \{(a, b) \in \Sigma \times \Sigma \mid a \Rightarrow^{dc,L} b \geq \theta_{\text{dep}} \wedge \text{quality}_L(dc_{a,b}) \geq \theta_{\text{con}}\}.$$

3. We build the set of dependency relations of the C-Net D as the union of unconditional- and conditional dependencies:

$$D = D_{\text{udr}} \cup D_{\text{cdr}}.$$

Some activities $s \in \Sigma$ might not have a predecessor or successor in the directed graph induced by D . Intuitively, each task in a process should have a cause (predecessor) and an effect (successor) [WR11], all tasks in the C-net should be connected. Therefore, we propose a new heuristic to enforce this: the *accepted-task-connected heuristic*²⁵. We repeatedly connect only those activities that are already part of the dependency graph using their best neighboring activities until all activities, except the artificial start activity a_i and the artificial end activity e_o , have a cause and an

²⁵The *all-task-connected heuristic* proposed in [WR11] could also be used. However, it has the shortcoming that all activities of the process are included, regardless of their occurrence frequency. This might obstruct the view on the conditional infrequent behavior.



(c) Dependency added by the accepted-task-connected heuristic to connect activity *Organize Ambulance*.

Figure 8.5: Dependency relations are discovered in three steps: (1) discovery of unconditional dependencies (D_{udr}), (2) discovery of extra conditional dependencies (D_{cdr}), and (3) adding extra missing dependencies through the accepted-task-connected heuristic (D_{atc}).

effect. Then, set D_{atc} of relations necessary to connect all accepted activities is:

$$\begin{aligned} D_{atc} = \{(a, b) \in (\Sigma \setminus \{a_i\}) \times (\Sigma \setminus \{a_o\}) \mid \\ (\nexists_x ((a, x) \in D \wedge \forall_y ((a \Rightarrow^{1,L} b) \geq (a \Rightarrow^{1,L} y)))) \\ \vee (\nexists_x ((x, b) \in D \wedge \forall_y ((a \Rightarrow^{1,L} b) \geq (y \Rightarrow^{1,L} b))))\}. \end{aligned}$$

We extend the dependency relations with the new relations, i.e., $D = D \cup D_{atc}$. There might be new, unconnected activities in D . Therefore, we *repeat adding the best neighboring activities* until set D_{atc} is empty.

Example 8.5 (Building the set of dependency relations). Figure 8.5 shows how the DHM discovers the dependency relations in three steps. Assume a noise-free event log that is generated by the hospital process depicted in Figure 8.2.

1. The standard (unconditional) dependency relations, which are shown in Figure 8.5a, are discovered. Infrequently observed relations are missing.
2. Four additional conditional dependencies are discovered based on dependency conditions obtained from the attributes in event log. All the infrequent process paths are discovered. Adding the dependencies leads to the inclusion of activity *Organize Ambulance*, which was previously not part of the model. However, activity *Organize Ambulance* is unconnected, i.e., no outgoing dependency relation was discovered.
3. Therefore, in the third step, the unconnected activity *Organize Ambulance* is connected by using the proposed accepted-tasks-connected heuristic by adding the dependency with the highest unconditional dependency score. In this case, the only successor activity a_o is added.

Discovering Input- and Output Bindings

The *input and output binding functions* of the C-net need to be discovered to determine the type of causal dependency between activities. Assume both $(a, b) \in D$ and $(a, c) \in D$ are discovered dependencies s.t. $b \neq c$. Then, we need to determine whether, after executing activity **a**, there is a choice between activities **b** and **c** or both activities should be executed in parallel. There are several methods to determine the input- and output bindings of a C-net. There are two categories:

- methods that use alignments to compute optimal bindings based on a cost function [ADA11b] and
- methods that use heuristics to determine likely bindings, which may be suboptimal [WA03; WAA06; WR11].

For the DHM, we use the heuristic proposed by the HM [WR11], since it is computationally more efficient than alignment techniques. We repeat its definition for the sake of completeness.

For the output binding function O of an activity $a \in \Sigma$, we need to determine which executions of $b \in \Sigma$ (with $(a, b) \in D$) were caused by an execution of activity a . The heuristic considers activity b to be caused by activity a only if it is the nearest activity that *may have caused* b . Any other activity x executed in between a and b should not be a possible cause of b , i. e., $(x, b) \notin D$.

Definition 8.7 (Set of activities caused by an event). Let $L = (E, \Sigma, \#, \mathcal{E})$ be an event log. Let $\sigma = \langle e_1, \dots, e_i, \dots, e_n \rangle \in \mathcal{E}$ be a trace. Let $D \subseteq \Sigma \times \Sigma$ be the set of discovered dependencies. We obtain the set of activities that were caused by an event e_i as:

$$\begin{aligned} actCaused(e_i) = \{b \in \Sigma &| \#_{act}(e_i) = a \\ &\wedge \exists_{i < j \leq n} \#_{act}(e_j) = b \wedge (a, b) \in D \\ &\wedge \forall_{i < k < j} (\#_{act}(e_k), b) \notin D\}. \end{aligned} \quad \diamond$$

We determine the frequency $|o|_{L,a} \in \mathbb{N}$ of an output binding $o \subseteq \Sigma$ for activity $a \in \Sigma$ in the event log L as:

$$|o|_{L,a} = |\{e \in E | \#_{act}(e) = a \wedge actCaused(e) = o\}|.$$

Then, we build the complete multi-set of output bindings with the most frequent bindings. Those bindings that fulfill the user-specified binding threshold θ_{bin} :

$$O(a) = \{o \subseteq \Sigma | \max_{\bar{o} \subseteq \Sigma} (|\bar{o}|_{L,a}) > 0 \wedge \frac{|o|_{L,a}}{\max_{\bar{o} \subseteq \Sigma} (|\bar{o}|_{L,a})} \geq \theta_{bin}\}.$$

The input binding function I is obtained by reversing the same approach.

Example 8.6 (Building the output binding function). Take an event log that consists only of the three exemplary traces shown in Table 8.1 and the dependency relations shown in Figure 8.5c. Assume that we want to determine the set of output bindings for activity *Diagnostic*, i. e., the set $O(\text{Diagnostic})$. We need to determine the frequencies $|o|_{L,a}$ for each possible output binding $o \subseteq \Sigma$. We consider only activities that are in a dependency relation with activity *Diagnostic*. Thus, the set of candidate output bindings is reduced to: $\{\{\text{Decide}\}, \{\text{Visit}\}, \{\text{Decide}, \text{Visit}\}\}$. We compute the frequencies of each output binding as:

$$\begin{aligned} |\{\text{Decide}\}|_{L,\text{Diagnostic}} &= 2 \\ |\{\text{Visit}\}|_{L,\text{Diagnostic}} &= 1 \\ |\{\text{Decide}, \text{Visit}\}|_{L,\text{Diagnostic}} &= 0. \end{aligned}$$

Thus, the candidate output binding {Decide} is activated twice, in σ_{h1} and in σ_{h3} , and candidate output binding {Visit} is activated once, in σ_{h2} . The output binding candidate {Decide, Visit} is never activated since according to the dependency relations both *Diagnostic* and *Visit* are possible causes for activity *Decide*. None of the events referring to an execution of activity *Diagnostic* can have caused both *Decide* and *Visit*. For example, in trace σ_{h2} , there is an *Visit* event (e_{h24}) in between the execution of *Diagnostic* (e_{h23}) and *Decide* (e_{h26}). Thus, we obtain set $actCaused(e_{h23}) = \{\text{Decide}\}$.

Combining the DHM with Existing Heuristics

Within the scope of this chapter, we do not elaborate on the various other heuristics defined by the HM [WR11], such as the long-distance heuristic, the length-two loop heuristic, and the relative-to-best heuristics. Those heuristics and further improvements proposed to the HM, e. g., those proposed by the Fodina miner [Bro14] can be used together with the DHM. However, the choice which heuristics to apply highly depends on the process at hand.

8.4 Extending Causal Nets with Multiple Perspectives

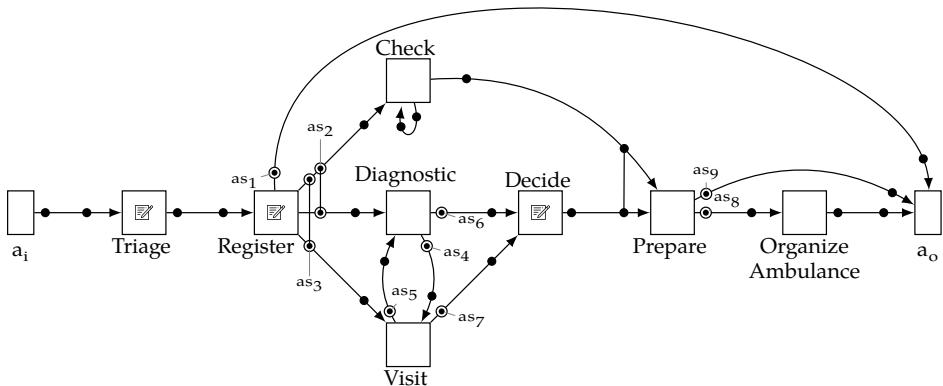
So far, we described how the DHM uses attributes values recorded in the event log to detect conditional dependencies. We use the information about the data-, resource, and time-perspective of a process that is stored in the event log to discover the control-flow of the model (i. e., the C-Net). However, the C-Net discovered by the DHM is agnostic to those other perspectives, i. e., only the control-flow of the process is defined. In this section, we propose an extension to the C-Net notation that captures the additional perspectives: **Data Causal Nets (DC-Nets)**. Also, we show how to extend the C-Net that is discovered by the DHM to a DC-Net, i. e., we describe the last Step 5 in Figure 8.3.

8.4.1 Data Causal Nets

We adopt a similar notation as used for DPNs and extend the C-Net with **attributes**, **write operations**, and **guard expressions**. However, DC-Nets remain to be defined using replay semantics (i. e., only complete sequences are part of the behavior).

Definition 8.8 (Data Causal Net (DC-Net)). Let $AS = \{X \subseteq \mathbb{P}(\Sigma) \mid X = \{\emptyset\} \vee \emptyset \notin X\}$ be a set of sets of activities. A DC-Net is a tuple $DC = (C, V_L, dom, wr, gd)$ where:

- $C = (\Sigma, a_i, a_o, D, I, O)$ is a C-Net;
- V_L is a set of attributes;

(a) The guarded output bindings as_1, \dots, as_9 are highlighted.

Binding	Guard expression	Activity	Variables
as_1	$color = \text{White}$	Triage	color
as_2	$color \neq \text{White} \wedge \text{resource} = \text{Alice}$	Register	resource
as_3	$color \neq \text{White} \wedge \text{resource} \neq \text{Alice}$		
as_4	$\text{resource} = \text{Alice}$	Decide	referral
as_5	$\text{resource} \neq \text{Alice}$		
as_6	$\text{resource} \neq \text{Alice}$		
as_7	$\text{resource} = \text{Alice}$		
as_8	$\text{referral} = \text{Tertiary}$		
as_9	$\text{referral} \neq \text{Tertiary}$		

(c) Variables written by activities.

(b) Guards assigned to output bindings.

Figure 8.6: Multiple perspectives of the simplified hospital process modeled as a DC-net.

- $dom : V_L \rightarrow \mathbb{P}(U)$ returns the (potentially infinite) domain for each attribute $v \in V_L$;
- $wr : \Sigma \rightarrow \mathbb{P}(V_L)$ returns the set of *write operations* for each activity;
- $gd : (\Sigma \times AS) \rightarrow \text{EXPR}(V_L)$ returns the boolean guard expression that is associated with each output binding of each activity, i.e., $dom(gd) = \{(a, as) \in \Sigma \times AS \mid as \in O(a)\}$.²⁶ \diamond

Attributes are updated by activities through write operations. Guard expressions need to be fulfilled before an output binding can be activated. Thus, it is possible to formulate multi-perspective constraints based on information encoded in attributes. We denote an output binding that is assigned a guard expression different from true as *guarded output binding*. We use the same function names as used by the DPN notation since the underlying concepts are similar and the applicable function should be clear from the context.

²⁶If an output binding $as \in O(a)$ is associated with no guard, we set $gd(a, as) = \text{true}$.

Example 8.7 (Simplified hospital process modeled as DC-Net). Figure 8.6 depicts a DC-Net that describes the behavior of the simplified hospital process. We extended the C-Net shown in Figure 8.4 with guarded output bindings for activities *Register*, *Diagnostic*, *Visit*, and *Prepare*. Moreover, we added *write operations* to activities *Triage*, *Register*, and *Decide*. Activities that write attributes are highlighted with a document symbol and Figure 8.6c lists the attributes that are written, e.g., the *Triage* activity writes the *color* attribute. *Guarded output bindings* are highlighted with circled dots and we assigned them identifiers and list the expression in Figure 8.6b. For example, there are three guarded output bindings for activity *Register*: as_1 , as_2 , and as_3 . The guards of the DC-Net correspond to the rules defined by the BPMN model shown in Figure 8.1.

Next, we formalize the execution semantics of a DC-net analogous to the semantics of a C-net by introducing its binding sequences, the *state* of a DC-Net, and its *language* through the set of valid binding sequences. Since, in a DC-Net, activities may write attributes, the binding sequences are extended with the current attribute assignment.

Definition 8.9 (Binding sequence of a DC-net). Let $DC = (C, V_L, dom, wr, gd)$ be a DC-Net. We denote the set of possible data-aware bindings of DC as:

$$DB = \{(a, w, as^I, as^O) \in (\Sigma \times \mathcal{U}_{dom}^L \times \mathbb{P}(\Sigma) \times \mathbb{P}(\Sigma)) \mid as^I \in I(a) \wedge as^O \in O(a)\}.$$

A binding sequence of a DC-net is a sequence of data-aware bindings: $\sigma \in DB^*$. \diamond

Definition 8.10 (State of a DC-Net). Let $DC = (C, V_L, dom, wr, gd)$ be a DC-Net. The *state* of a DC-Net $ST_{DC} = (\alpha, OBL) \in \mathcal{U}_{dom}^L \times \mathbb{B}(\Sigma \times \Sigma)$ consists of the current attribute assignment α and a multiset of *pending obligations* OBL . We define function $state_{DC} : DB^* \rightarrow ST_{DC}$ that returns the state after executing binding sequence $state_{DC}(\sigma)$ as:

- $state_{DC}(\langle \rangle) = (\emptyset, [])$
- $state_{DC}(\sigma \cdot (a, w, as^I, as^O)) = (\alpha \oplus w, OBL \setminus (as^I \times \{a\})) \uplus (\{a\} \times as^O)$ s.t. $(\alpha, OBL) = state_{DC}(\sigma)$

for any binding sequences $\sigma \cdot (a, as^I, as^O) \in DB^*$. \diamond

Both binding sequence and the state of DC-Net keep track of the attribute assignment by activities. As in a C-Net the pending obligations can be seen as tokens, which need to be consumed by subsequent activities. However, pending obligations may be only consumed if the guard expressions on the guarded output bindings are fulfilled wrt. to the current attribute assignment of the state.

Definition 8.11 (Valid binding sequences of a DC-Net). Let $DC = (C, V_L, dom, wr, gd)$ be a DC-Net. Let $\sigma = \langle (a_1, w_1, as_1^I, as_1^O), \dots, (a_n, w_n, as_n^I, as_n^O) \rangle \in DB^*$ be a binding sequence. Sequence σ is a valid binding sequence of DC if and only if:

1. $a_0 = a_i \wedge \forall_{1 < j < n} (a_j \in (\Sigma \setminus \{a_i, a_o\})) \wedge a_n = a_o$;
2. there are no pending obligations, i.e., $state_{DC}(\sigma) = (\alpha, []);$ and
3. for any prefix $\langle (a_1, w_1, as_1^I, as_1^O), \dots, (a_j, w_j, as_j^I, as_j^O) \rangle = \sigma' \cdot (a_j, w_j, as_j^I, as_j^O) \in pref(\sigma)$ and $state_{DC}(\sigma') = (\alpha, OBL)$ we require that:
 - $(as_j^I \times \{as_j^O\}) \leq OBL$, i.e., bindings may only remove pending obligations for the current activity as_j ;
 - $dom(w_j) = wr(a_j)$, i.e., the activity writes the required attributes;
 - $\forall_{v \in dom(w_j)} (w(v) \in dom(v))$, i.e., the written attribute values are valid according to the domain; and
 - $eval(gd(a_j, as_j^O), \alpha) = \text{true}$, i.e., the guard expression is fulfilled.

BS_{DC} is the set of valid binding sequences of the DC-net DC . ◊

Example 8.8 (Valid binding sequences). Take the DC-Net shown in Figure 8.6. We abbreviate activities as follows: a_{tri} is the activity *Triage*, a_{reg} is the activity *Register* etc.. The following binding sequence corresponds to trace σ_{h1} (cf., Table 8.1):

$$\begin{aligned} \sigma = & \langle (a_i, \emptyset, \emptyset, \{a_{tri}\}), \\ & (a_{tri}, (\text{color} \mapsto \text{Red}), \{a_i\}, \{a_{reg}\}), \\ & (a_{reg}, (\text{resource} \mapsto \text{Joe}), \{a_{tri}\}, \{a_{che}, a_{vis}\}), \\ & (a_{che}, \emptyset, \{a_{che}\}, \{a_{che}\}), (a_{che}, \emptyset, \{a_{che}\}, \{a_{che}\}), (a_{che}, \emptyset, \{a_{che}\}, \{a_{pre}\}), \\ & (a_{vis}, \emptyset, \{a_{reg}\}, \{a_{dia}\}), (a_{dia}, \emptyset, \{a_{vis}\}, \{a_{dec}\}), \\ & (a_{dec}, (\text{referral} \mapsto \text{Ward}), \{a_{dia}\}, \{a_{pre}\}), \\ & (a_{pre}, \emptyset, \{a_{che}, a_{dec}\}, \{a_o\}), \\ & (a_o, \emptyset, \emptyset, \emptyset) \rangle. \end{aligned}$$

Binding sequence σ is a valid binding sequence of the DC-Net, i.e., $\sigma \in BS_{DC}$ since: (1) σ starts with a_i and ends with a_o , (2) there are no pending obligations in state $state_{DC}(\sigma)$, (3) all prescribed write operations are present, and (4) all guard expressions are fulfilled.

We abbreviate attributes as follows: **res** refers to *resource*, **ref** refers to *referral*, and **col** refers to *color*. The state of the DC-Net changes as follows while replaying

binding sequence σ . All conditions are fulfilled in every step:

$$\begin{aligned}
 \sigma_1 &= \langle \rangle, & ST_{DC}(\sigma_1) &= (\emptyset, []) \\
 \sigma_2 &= \sigma_1 \cdot (\underline{a_i}, \emptyset, \emptyset, \{a_{tri}\}), & ST_{DC}(\sigma_2) &= (\emptyset, [(a_i, a_{tri})]) \\
 \sigma_3 &= \sigma_2 \cdot (\underline{a_{tri}}, (col \mapsto Red), \{a_i\}, \{a_{reg}\}), & ST_{DC}(\sigma_3) &= ((col \mapsto Red), [(a_{tri}, a_{reg})]) \\
 \sigma_4 &= \sigma_3 \cdot (a_{reg}, (res \mapsto Joe), (a_{tri}), (a_{che}, a_{vis})), & ST_{DC}(\sigma_4) &= ((col \mapsto Red), (res \mapsto Joe), [(a_{reg}, a_{che}), (a_{reg}, a_{vis})]) \\
 \sigma_5 &= \sigma_4 \cdot (a_{che}, \emptyset, \{a_{che}\}, \{a_{che}\}), & ST_{DC}(\sigma_5) &= ((col \mapsto Red), (res \mapsto Joe), [(a_{che}, a_{che}), (a_{reg}, a_{vis})]) \\
 \sigma_6 &= \sigma_5 \cdot (\underline{a_{che}}, \emptyset, \{a_{che}\}, \{a_{che}\}), & ST_{DC}(\sigma_6) &= ((col \mapsto Red), (res \mapsto Joe), [(a_{che}, a_{che}), (a_{reg}, a_{vis})]) \\
 \sigma_7 &= \sigma_6 \cdot (\underline{a_{che}}, \emptyset, \{a_{che}\}, \{a_{pre}\}), & ST_{DC}(\sigma_7) &= ((col \mapsto Red), (res \mapsto Joe), [(a_{che}, a_{pre}), (a_{reg}, a_{vis})]) \\
 \sigma_8 &= \sigma_7 \cdot (a_{vis}, \emptyset, \{a_{reg}\}, \{a_{dia}\}), & ST_{DC}(\sigma_8) &= ((col \mapsto Red), (res \mapsto Joe), [(a_{che}, a_{pre}), (a_{vis}, a_{dia})]) \\
 \sigma_9 &= \sigma_8 \cdot (\underline{a_{dia}}, \emptyset, \{a_{vis}\}, \{a_{dec}\}), & ST_{DC}(\sigma_9) &= ((col \mapsto Red), (res \mapsto Joe), [(a_{che}, a_{pre}), (a_{vis}, a_{dec})]) \\
 \sigma_{10} &= \sigma_9 \cdot (\underline{a_{dec}}, (ref \mapsto Ward), \{a_{dia}\}, \{a_{pre}\}), & ST_{DC}(\sigma_{10}) &= ((col \mapsto Red), (res \mapsto Joe), (ref \mapsto Ward), \\
 &&&\quad [(a_{che}, a_{pre}), (a_{dec}, a_{pre})]) \\
 \sigma_{11} &= \sigma_{10} \cdot (\underline{a_{pre}}, \emptyset, \{a_{che}, a_{dec}\}, \{a_o\}), & ST_{DC}(\sigma_{11}) &= ((col \mapsto Red), (res \mapsto Joe), (ref \mapsto Ward), [(a_{pre}, a_o)]) \\
 \sigma_{12} &= \sigma_{11} \cdot (\underline{a_o}, \emptyset, \{a_{pre}\}, \emptyset), & ST_{DC}(\sigma_{12}) &= ((col \mapsto Red), (res \mapsto Joe), (ref \mapsto Ward), [])
 \end{aligned}$$

We highlighted guarded output bindings such as $as_3 = \{a_{che}, a_{vis}\}$, which was added in σ_4 . The guard of binding as_3 needs to be fulfilled in state $ST_{DC}(\sigma_4)$, i.e., $gd(Register, as_3) \leftarrow (color \neq \text{Alice} \wedge resource \neq Alice)$ needs to hold. The attribute assignment in state $ST_{DC}(\sigma_4)$ is $(color \mapsto Red, resource \mapsto Joe)$ and, thus, the guard is fulfilled.

We can, now, express the behavior of a DC-Net using the trace set notation that we introduced in Section 3.1, i.e., we can use DC-Nets as input to other presented methods that require trace sets as input. For example, alignments and the fitness and precision of DC-Net can be defined. The C-Net notation does not distinguish between observable activities and process transitions as well as between attributes and variables. Therefore, we assume $\Sigma = T$ and $V_P = V_L$ since a separate definition of a labeling function is not required. To obtain the trace set of a DC-Net, we take the sequence of activity executions and write variable assignments and remove the introduced artificial activities a_i and a_o since they do not represent real work in the process.

Definition 8.12 (Trace set of a DC-Net). Let BS_{DC} be the set of valid binding sequences of the DC-net $DC = (C, V_L, dom, wr, gd)$. Let $T = \Sigma$ be the set of process transitions. Let $V_P = V_L$ be the set of process variables. Let $PS = (T \times U_{dom}^P)$ be the

set of all possible process steps. We define the trace set of DC-Net as:

$$\begin{aligned} TS_{DC} = \{ & \langle (t_1, w_1), \dots, (t_n, w_n) \rangle \in PS^* \mid \exists_{\sigma \in BS_{DC}} (\sigma = \langle (a_i, w_i, as_i^I, as_i^O), \\ & (t_1, w_1, as_1^I, as_1^O), \dots, \\ & (t_n, w_n, as_n^I, as_n^O), \\ & (a_o, w_o, as_o^I, as_o^O) \rangle) \}. \quad \diamond \end{aligned}$$

We can also convert the DC-Net to a DPN that over-approximates its behavior. We use the transformation from C-Nets to Petri nets that is proposed by Aalst et al. in [AAD11] and extend the resulting Petri net with the write operations and guard expressions of the DC-Net. This conversion enables us to leverage existing methods for DPNs and Petri nets, such as the balanced alignment method presented in Chapter 5. Figure 8.7 shows such a conversion from the hospital process DC-Net to a DPN. Each input- and output binding is transformed to an invisible routing transition. Guards on output bindings are transformed to guards on the corresponding invisible transitions.

8.4.2 Discovering DC-Nets

We extend the C-Net by discovering the write operations and guarded output bindings of a DC-Net. We already discovered conditional dependencies between activities. For example, the dependency relation between activities *Prepare* and *Organize Ambulance* was discovered to be associated with the condition: *referral* = *Tertiary*. However, those conditions were discovered on the level of dependency relations (i. e., the edges between *Prepare* and *Organize Ambulance*) and not on the level of output bindings. One dependency relation may be element of several output bindings. Therefore, we cannot directly use the dependency conditions that were discovered in Section 8.3.2. Here, we make use of existing decision mining methods, such as the one later presented in Chapter 10 or the one presented in [LA13b]. Since we will elaborate on decision mining method in more detail in Chapter 10, we only introduce the idea briefly.

The main challenge for all decision mining methods is to *build a set of observation instances* based on the event log, i. e., for which attribute values a certain alternative was taken. Here, the alternatives are the different output bindings for an activity (e. g., $O(\text{Prepare}) = \{\{\text{Organize Ambulance}\}, \{a_o\}\}$). Our goal is to find a good guard function for the DC-Net, i. e., good guard expressions for the output bindings of each activity based on the attribute values recorded beforehand. For example, we would like to infer that the guard expression for the output binding $\{\text{Organize Ambulance}\}$ of activity *Prepare* is: (*referral* = *Tertiary*), i. e., $gd(\text{Prepare}, \{\text{Organize Ambulance}\}) \leftarrow (\text{referral} = \text{Tertiary})$. The set of observation instances for an activity of a DC-Net based on an event log is obtained as follows.

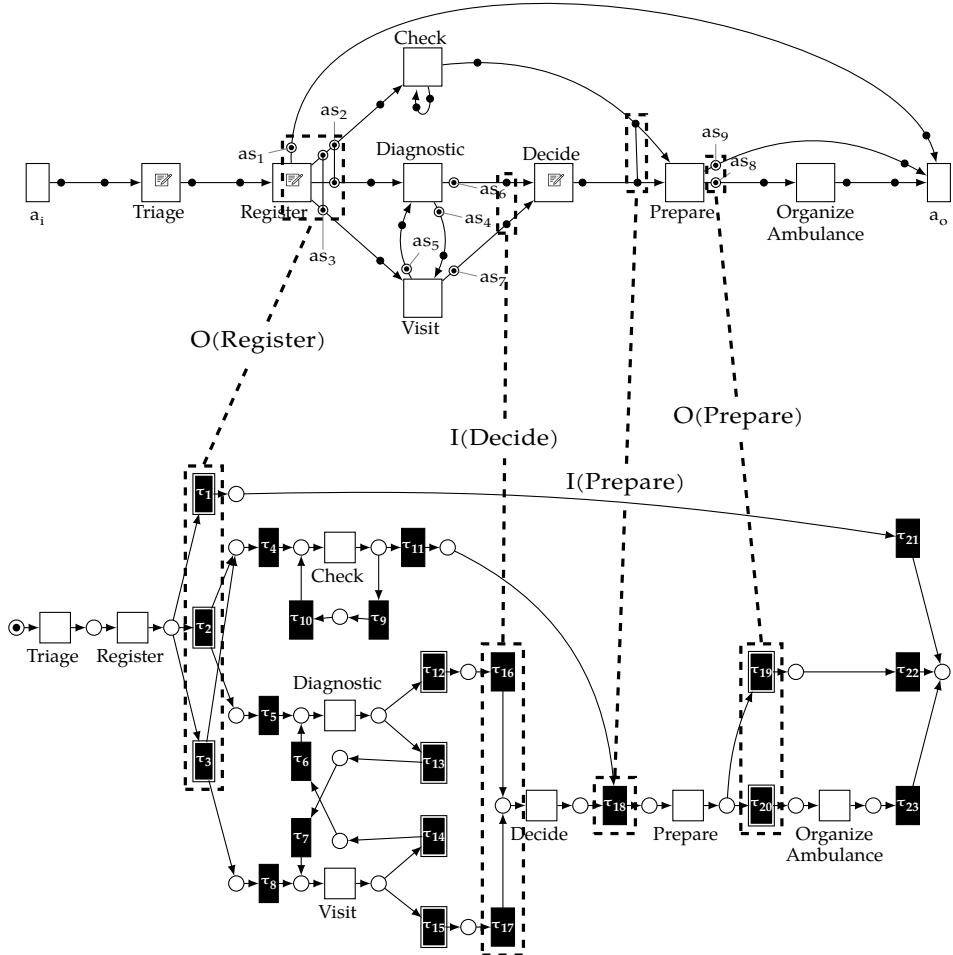


Figure 8.7: Conversion from the DC-Net of the hospital process to a DPN that over-approximates its behavior. Each input- and output binding is transformed to an invisible routing transition. Guards on output bindings are transformed to guards on the corresponding invisible transitions. We simplified the DPN to improve the legibility of the figure by removing the introduced invisible transitions for activities with only one input- or output binding. Those transitions are not strictly necessary.

Definition 8.13 (Observation instances for a C-Net activity). Let $L = (E, \Sigma, \#, \mathcal{E})$ be an event log. Let $C = (\Sigma, a_i, a_o, D, I, O)$ be a C-Net. Let $AS = \{X \subseteq \mathbb{P}(\Sigma) \mid X = \{\emptyset\} \vee \emptyset \notin X\}$ be a set of sets of activities. Function $OI_{AS,L} : \Sigma \rightarrow \mathbb{B}(\mathcal{U}_{dom}^L \times AS)$ returns the multi-set of observation instances for each activity of the C-Net:

$$OI_{AS,L}(a) = \bigcup_{e \in E \mid \#_{activity}(e)=a} [(latest(e), actCaused(e))] \quad \diamond$$

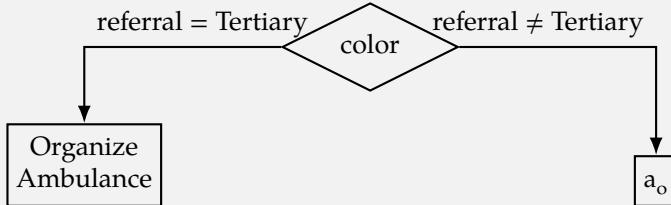
Having determined the observation instances, we use the techniques presented in [LA13b] and Chapter 10 to discover guards for the output bindings, i. e., we estimate the guard function gd . Here, we do not elaborate further on how guards are obtained since this is discussed in-depth in Chapter 10.

As last step, we discover the write operations of the DC-Net. We assume that each activity writes the attributes that have been observed to be written in the event log for that activity. We retain the attributes that are used in the guards of the DC-Net since not all attributes may be important.

Example 8.9 (Discovering guarded output bindings). Assume that we want to discover the guards for the output bindings of activity *Prepare*. We collect a set of observation instances based on the three exemplary traces:

$$OI_{AS,L}(\text{Prepare}) = [((color \mapsto \text{Red}, \text{resource} \mapsto \text{Joe}, \text{referral} \mapsto \text{Ward}), \{a_o\})^2, ((color \mapsto \text{Red}, \text{resource} \mapsto \text{Alice}, \text{referral} \mapsto \text{Ward}), \{\text{Organize Ambulance}\})].$$

Based on the observation instance, we can employ, e. g., a C4.5 decision classifier to build the following decision tree:



Then, the obtained guard function returns the following guards for activity *Prepare*:

$$\begin{aligned} gd(\text{Prepare}, \{\text{Organize Ambulance}\}) &\leftarrow (\text{referral} = \text{Tertiary}) \\ gd(\text{Prepare}, \{a_o\}) &\leftarrow (\text{referral} \neq \text{Tertiary}) \end{aligned}$$

The leafs of the decision tree are converted to terms of the guard expression. The exact conversion from a C4.5 decision tree to as set of guard expression is explained more detailed in Chapter 10.

8.5 Evaluation

We evaluated the DHM based on both real-life and synthetic data sets. To evaluate the method, we implemented the DHM in the DataAwareCNetMiner package of the open-source framework ProM 6.7. In Section 11.1 we describe the implementation, which provides an *interactive tool*, which allows to quickly discover C-nets for different parameter settings and to explore the discovered data dependencies.

In the interest of a clear separation between the more theoretical contributions of this thesis and their applications to case studies, we describe the evaluation on real-life data sets separately as part of the presentation of case studies in Sections 12.4, 13.4 and 15.4.

The experiments for the synthetic data show that the DHM is resilient to a certain degree of randomly injected noise, which is not characterized by data conditions. It rediscovers the original model, whereas earlier techniques either show too much, or too little behavior.

8.5.1 Event Log and Methods

We generated an event log with 100,000 traces and approximately 1,000,000 events by simulating the process model shown in Figure 8.1 using CPN Tools [JKW07]. Three data attributes are recorded in the event log: *color*, *resource*, and *referral*. We adjust the frequency distributions of the recorded attributes values such that paths Ⓐ, Ⓑ, and Ⓒ in model Figure 8.1 are recorded infrequently. Specifically, only:

- 1.4% of the traces have an event setting the attribute *color* = White,
- 33.4% of the traces have an event setting the attribute *resource* = Alice, and
- 1.7% of the traces have an event setting the attribute *referral* = Tertiary.

We checked that the generated event log is fully fitting the hospital process model shown in Figure 8.1. Our method (DHM) was compared with the following two methods: the Heuristics Miner [WR11] with frequency filtering based on θ_{obs} (HMF) and the Heuristics Miner [WR11] without frequency filtering (HMA). For all three methods, we used the following standard parameters: $\theta_{obs} = 0.1$ (0.0 for HMA), $\theta_{dep} = 0.9$, $\theta_{bin} = 0.1$, $\theta_{con} = 0.5$, and the *accepted-task-connected heuristic*. To discover dependency condition, we used C4.5 as classifier and estimated its performance with 10 times 10-fold cross validation to obtain the quality score.

8.5.2 Experimental Design

The experiment aims to assess the *efficiency* and *effectiveness*, in term of noise filtering capabilities, of our method. For this purpose, we injected three different types of noise into the event log by randomly:

- *adding* one additional event to an increasing number of traces,
- *removing* one additional event to an increasing number of traces, and

- swapping one event with another randomly selected event in an increasing number of traces.

Then, we compared the discovered dependency relations with those of the C-net reference model (Figure 8.4) in terms of graph edit distance (GED) [DDG09]. GED measures the difference between two C-Nets based on the number of edit operations required to transform one C-Net into the other C-Net based on the two operations: adding or removing an activity and adding or removing a dependency relation. Note that we ignore the difference in bindings for this experiment since we are solely interested in discovering the correct dependency relations.

For this evaluation, we did not use fitness, precision, or behavioral comparison measures as those measures would not be applicable in this setting. Fitness and precision do not measure the quality of the model with regard to a reference model (gold standard). Moreover, when the discovered models are not sound (e. g., having a deadlock), the behavior may be undefined even when the model is close to the original model and provides insights. Behavioral measures would also fail to distinguish the difference between the data-dependent re-sequencing of activities (pattern © in Figure 8.4) and simple parallelism. For example, both in Figure 8.2b and in Figure 8.2a activities Visit and Diagnostics can be interleaved in any order.

8.5.3 Results

Regarding the *efficiency* of the method, we could discover all C-Nets in less than 3 seconds of computation time using 2 GB of memory on a standard laptop. Most of the computation time was spent in the training and validation of the C4.5 classifier, which we use to discover data conditions. Since we employed cross validation, the training procedure was repeated 10 times each time for 10 folds. Thus, we consider that the DHM can be used in a real-life settings with large event logs.

Next, we investigated the *effectiveness* of our method, i. e., whether the DHM is able to rediscover the infrequent conditional relations and distinguish them from random noise. Without any noise, our method was able to rediscover the relations ®, ®, and ©, i. e., the highlighted edges in Figure 8.4. The original rules Color = White and Referral = Tertiary were discovered for relations ® and ©. For path ©, two rules were discovered: Nurse = Alice for the edge from Diagnostics to Visit and Nurse ≠ Alice for the edge from Visit to Diagnostics. Thus, our method discovered the conditional re-sequencing of activities Visit and Diagnostics, whereas the standard Heuristics Miner (cf., the model in Figure 8.2b) considered both activities as parallel. Moreover, when extended the C-Net to a DC-Net, we could discover the guard expressions shown in Figure 8.6.

We added increasing levels of noise to the event log and compared the discovered models with the reference model. Figure 8.8 shows the result of the GED measurement for noise levels ranging from 0% to 100% (i. e., up to 100% of the traces are injected with noise) and the three considered types of noise.

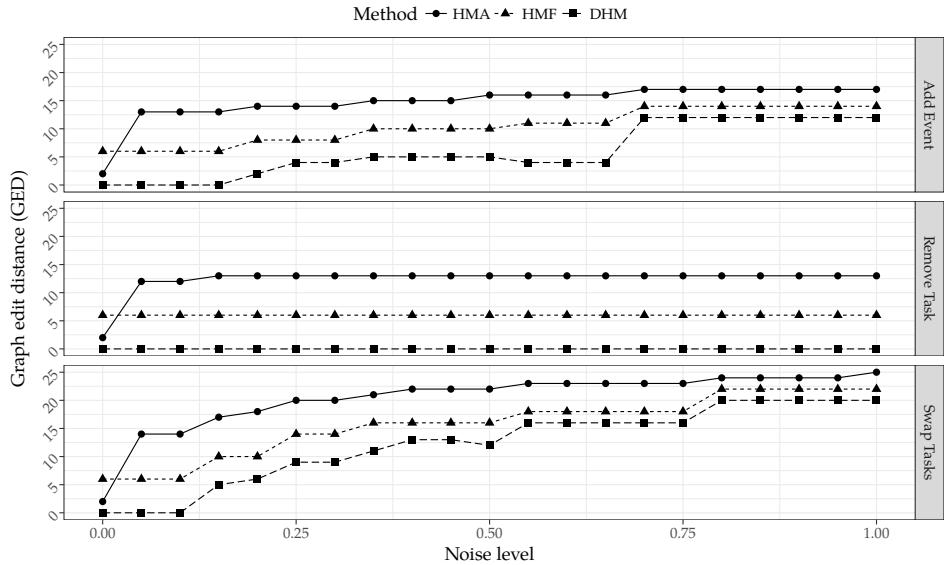


Figure 8.8: GEDs between the relations discovered by the compared methods and relations of the reference model for varying noise levels and noise types.

Add Event. Our method (DHM) handles the added noise well until 15% of the traces were modified. Until 65% of the traces were affected by noise the DHM returns a model that is closer to the reference model than the one returned by the standard Heuristics Miner with noise filtering (HMF). The HMF method is rather insensitive to the injected noise. However, it fails to discover the reference model even without noise, as shown in Figure 8.8 and evident in Figure 8.8. When lowering the observation frequency threshold to 0.0 (method HMA), already 5% of injected noise affect the discovery and undesirable dependencies appear.

Remove Event. Interestingly, neither the DHM nor the HMF method seem to be affected by that kind of noise. However, as before the HMF method does not discover the infrequent relations for the event log without noise. This can be explained by the distribution of the different events in the event log. Since activity *Check* can be repeated, it occurs more frequently in the event log than other activities. Therefore, it is more likely that an event recorded the *Check* activity is removed. Removing one event of the many events that refer to the *Check* activity is unlikely to have a large effect on the mined C-net.

Swap Event. Swapping two events randomly in an increasing number of traces quickly challenges both the classical Heuristics Miner method (HMF) as well as our DHM. The reference model is rediscovered for up to 10% noise. Then, as the classical HMF method also our method get affected by the injected noise. This can be expected because by swapping events the recorded attribute values might also be swapped, which makes it more unlikely to discover the correct data conditions.

In all of the comparisons, we did not include the IM in Figure 8.8, as it returns models with a different structure than C-nets. Therefore, a direct comparison would not be fair. However, the models returned by the IM already fail to rediscover the original model for an event log with 5% noise as shown in Figure 8.2a.

8.6 Related Work

Most related to the work presented in this chapter are approaches for *noise filtering* and *multi-perspective process discovery*.

8.6.1 Noise Filtering Techniques

Early techniques for process discovery often assumed noise-free event logs, e.g., the Alpha algorithm [AWM04] and the region-based approaches [CCK08]. These techniques are of limited use in real-life settings. Most of the more recent process discovery methods support some type of noise filtering [Wee+12].

These noise-filtering methods are often based on frequencies and ignore the event payload (i.e., attributes). All those methods fail to distinguish infrequent behavior that is characterized by deterministic rules from random noise. For example, the Fuzzy Miner [GA07], the Heuristics Miner [WR11], the Inductive Miner [LFA13], the Constructs Competition Miner [Red+14], methods based on maximal-pattern mining proposed in [LYC15], and genetic methods such as the Genetic Miner [MWA07] and the Evolutionary Tree Miner [BDA12] are all based on counting frequencies when distinguishing noise from regular process behavior. Further examples are a method based on injecting negative events (i.e., events that are not supposed to occur) that discover declarative models [Goe+09], a method that uses negative events to improve existing discovery methods [PCB15], a method that includes prior knowledge on probability distributions [Rem+13], and a method that discovers a models expressed in a probabilistic logic [BRL16].

Dedicated noise filtering methods have also been proposed, e.g., a method based on automata [CLH16] and a method based on outlier detection [Ghi+08]. Again, both methods are based on the control-flow perspective.

All of the discussed methods are tailored towards filtering infrequent noise based on the control-flow perspective. Thus, those methods are not able to distinguish conditional infrequent behavior from random infrequent behavior.

8.6.2 Multi-perspective Discovery

Decision mining methods have been proposed to discover conditional behavior of processes. Smedt et al. proposed to distinguish two types of decision mining techniques [De +17a; De +17b]:

- (A) decision-annotated process mining, i. e., first discover a suitable control-flow and, then, annotate the model with decision rules and
- (B) decision-aware control-flow, i. e., control-flow and decision are discovered together in an holistic manner.²⁷

We consider our method to be of type (B). The dependency conditions discovered with decision mining techniques based on the event log influence the control-flow of the discovered C-Net. Both control-flow and data are discovered in a holistic manner. However, differently to [De +17b] we do not integrate a full decision model (e.g., a DMN decision requirement diagram [DMN16]) into the model but only annotate dependencies with rules.

There are several existing process discovery methods that use a staged approach (i. e., type A) to discover data-aware process models. First, the control-flow is discovered and, then, the process model is enhancement with other perspectives. Examples of such approaches are the following decision mining techniques [Bat+15; BBW16; LA13b; LDG13; Roz+09] and the proposed method to discover overlapping rules in Chapter 10. However, these staged approaches do not leverage the full potential of other perspectives on the process. Process discovery method should consider multiple perspectives together: data- and control-flow perspective need to be discovered together.

There are also some proposals for method of type B, which are related to our work. For example, recent work on declarative process discovery [Sch+16b] considers the data perspective first. However, similar to association rule mining, sets of rules rather than full process models are returned. Moreover, work on the discovery of artifact lifecycles [PFD15] can be considered to follow a data-first approach, i. e., business artifacts and their lifecycles are considered to be the starting point for process discovery. Still, the role of noise and infrequent behavior is not discussed. The approach presented in [De +17b] also goes beyond discovering the decision for an existing control-flow model. However, in [De +17b] a decision requirement diagram [DMN16] instead of a control-flow model with integrated decision logic is discovered based on information recorded in the event log. Finally, [Aa+16] describes a method that aims to separate the decision logic, which is often implicitly encoded in process model, from the control-flow. Thus, the method considers the integration of decision logic with control-flow. However, the method does not use an event log as source of information about the process.

²⁷The two types of decision mining are denoted as Q3 and Q4 respectively in the quadrant presented in [De +17a; De +17b]. However, Q1 (control-flow discovery) and Q2 (data mining without a process model) are not applicable in our setting.

8.7 Conclusion

In this chapter, we presented the Data-aware Heuristic Miner (DHM), a *process discovery* method that *reveals conditional infrequent behavior* from event logs.

8.7.1 Contribution

The DHM distinguishes undesired noise from infrequent behavior that can be characterized by conditions over the data attributes of the event log. This is the *first approach that uses both event labels and data attributes when discovering the control-flow*. Data- and control-flow are learned together. We employ classification techniques to discover conditional dependencies based on the attribute values recorded in the event log. Then, we use those conditional dependencies to discover C-Nets based upon the ideas proposed by the Heuristic Miner [WR11]. The discovered C-Nets are annotated with information on the data perspective, i. e., the discovered classification rules. Moreover, we extend the C-Net notation, which is still focused on the control-flow perspective, to the data-aware DC-Net notation. By using the DC-Net notation existing methods for decision mining can be applied on the C-Net discovered by the DHM. We systematically evaluated the DHM by using a synthetic data set. The evaluation showed that our approach can efficiently handle large event logs with several attributes and distinguish between typical levels of random noise and conditional infrequent behavior.

We implemented the DHM as an interactive tool in the open-source process mining framework ProM. The tool is described in Section 11.1. Later in Sections 12.4, 13.4 and 15.4, we describe how we used the interactive DHM (iDHM) in three case studies. In all three case studies, we revealed interesting infrequent conditional behavior from real-life event logs.

8.7.2 Limitations

We acknowledge that there are some limitations to our method.

- We only consider conditional directly-follows dependencies. Like most process mining approaches, our method requires sufficiently large event logs. Small event logs might, by chance, not contain all directly-follows relations. Moreover, more complex patterns of conditional infrequent behavior, e. g., longer sequences or sub-processes, cannot be discovered.
- There is a risk that the C-Nets returned are unsound [AS11] since our method is based on the Heuristics Miner, which does not guarantee soundness.
- We only tested the noise filtering capabilities of our method based on the example hospital process. Therefore, only limited claims on the general applicability of the noise filtering capabilities of the method can be made.

8.7.3 Future Work

There are several directions for future work that are worthwhile exploring.

- The idea could be extended from directly-follows relations to more complex patterns of conditional behavior (e.g., conditional long-term dependencies). Our method successfully reveals data dependencies based on directly-follows relations, but dependencies that cannot be captured by directly-follows relations might be missed.
- Recent research shows that it is possible to structure models discovered by the Heuristics Miner to prevent the discovery of unsound models [Aug+16]. Since our method returns the same type of models as discovered by the Heuristic Miner (i.e., C-Nets), it would be interesting to apply this structuring method together with our method to reveal infrequent conditional behavior.
- Our method supports the time perspective when it is encoded as data attribute, e.g., conditional relations that appear for cases with a high throughput time. However, the time perspective has particular characteristics that warrant further investigation, e.g., time is monotonically increasing within a process instance and the duration of process activities is usually determined by several correlated events recording life-cycle transitions.

9 Guided Multi-perspective Process Discovery

Most process mining techniques assume that recorded events correspond to meaningful activities in the instances of a process (i. e., cases). The information about recorded executions of activities can then be used, e. g., to *discover* models describing the observed behavior or to check *conformance* with respect to existing regulations, procedures, and process documentation. The ability to identify executions of activities based on events and discover recognizable models is crucial for any process mining technique. Events that do not directly correspond to *high-level activities* recognizable for process workers are unsuitable for process analytics since their semantics are not clear to domain experts. Process models discovered based on such low-level events are often incomprehensible to stakeholders. Process discovery methods need to consider the function perspectives when dealing with such event logs. Events recording the execution of *low-level activities* should be abstracted to activity instances of *high-level activities* prior to applying process discovery methods.

Table 9.1: Low-level and high-level activities on the type and instance level. The goal of this chapter is to identify the instances of high-level activities (i. e., activity instances) from events recording instances of low-level activities.

abstraction level	low	high
type level	low-level activity	high-level activity
instance level	event	activity instance

As shown in Table 9.1, we assume that the events in the input event log are recorded at a low abstraction level, i. e., instances of low-level activities are recorded in a low-level event log. Our goal is to identify executions of high-level activities, which we denote as *activity instances*, from the event log. Before presenting our proposal for an event abstraction method, we motivate the need for event abstraction with an example.

9.1 Motivation for Event Abstraction

We introduce *event abstraction* using a motivating example. Here, we do not use the running example hospital process example since that process is less suited for the challenges we want to illustrate. Instead, we use the actual hospital system of a Norwegian hospital, which will be described in more detail as part of the case study in Chapter 14. The hospital uses digital whiteboards to manage the work of nurses on the hospital ward. The whiteboards record the work of nurses at fine granularity: each action of the nurse is registered. Moreover, the system is connected to the call signal system of the hospital. This system records events when a patient presses an alarm button and when nurses clear the alarm state by pressing a different button.

The upper part of Figure 9.1 depicts four high-level activity instances that were carried out by nurses of the hospital for one patient: *Shift*, *Alarm*, *Alarm*, and *Handover*. The second instance of *Alarm* and *Handover* are carried out in parallel. The lower part of Figure 9.1 shows the events that are actually recorded by the digital whiteboard system while nurses carried out the four high-level activities. As depicted in Figure 9.1 one high-level activity instance may result in multiple *low-level events* being recorded. For example, the first activity instance *Shift* resulted in three low-level events being recorded. Groups of low-level events correspond to instances of high-level activities. Note that there are different sequences of events that corresponding to an instance of high-level activity *Shift*. Also note that low-level events can be mapped to different activities instances (e.g., *CallSignal0* is both recorded instances of *Shift* and *Alarm*).

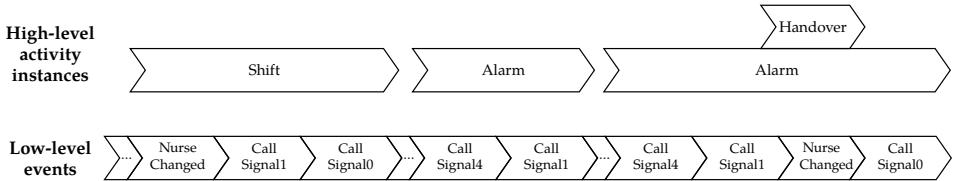


Figure 9.1: Mapping between events recording the occurrence of low-level activities and instances of the actual high-level activities that were executed for a process instance.

As illustrated by the example, events recorded by information systems often do not match instances of high-level activities [BMW14; GRA10], i.e., there are functional dependencies between events that are not uncovered by process discovery methods. Generally, there can be an n:m-relation between recorded low-level and high-level activities both on the type and instance level [BMW14; GRA10], i.e., one *high-level activity instance* may result in multiple *low-level events* being recorded and, vice versa, one such low-level event may relate to multiple high-level activity instances.

When applying process discovery methods on low-level event logs, semantically related activities are not presented as such. Grouping those related activities to-

gether into higher level activities facilitates model comprehension [RMD11]. Thus, *event abstraction*, the grouping of events to recognizable activities on a higher abstraction level, can help to guide process discovery methods towards discovering a process model that can be understood by stakeholders and is more useful for answering process questions. This chapter reports on an event abstraction method, which groups sequences of low-level events to high-level activity instances.

There are several challenges for event abstraction methods [Bai15]:

- shared functionality, i. e., a low-level activity is carried out as part of the work for more than one high-level activity;
- noise, i. e., missing and additional unexpected low-level events are recorded in the event log;
- repetition, i. e., high-level activities reoccur; and
- parallelism, i. e., two or more high-level activities are carried out in parallel.

The goal is to obtain the original mapping between the events recording low-level activities and the actual high-level activity instances, i. e., we want to determine the upper part of Figure 9.1 based on the low-level events in the lower part. We now introduce the motivational example in more detail in Table 9.2 and Example 9.1.

Table 9.2: Excerpt of an example trace $\sigma_{wb} \in \mathcal{E}_L$ from an log-level event log L_L that contains low-level events recorded by low-level activities executed on an electronic whiteboard. The last two columns *hl activity* and *hl instance* indicate which instance of which high-level activity caused the recording of a low-level event.

low-level events σ_{wb}				high-level activity instances	
id	ll activity	time	nurse	hl activity	hl instance
...
e ₁₂	NurseChanged	122	Nancy	Shift	3
e ₁₃	CallSignal1	122		Shift	3
e ₁₄	CallSignal0	124		Shift	3
...
e ₂₀	CallSignal4	185		Alarm	6
e ₂₁	CallSignal1	194		Alarm	6
...
e ₃₀	CallSignal4	310		Alarm	11
e ₃₁	CallSignal1	311		Alarm	11
e ₃₂	NurseChanged	312	Nathan	Handover	12
e ₃₃	CallSignal0	315		Alarm	11

Example 9.1 (Low-level event log and high-level activity instances). The first four columns of Table 9.2 show an excerpt of a trace $\sigma_{wb} \in \mathcal{E}_L$ obtained from a low-level event log $L_L = (E_L, \Sigma_L, \#^L, \mathcal{E}_L)$ that is recorded by a digital whiteboard, which supports the work of nurses in a hospital. We omitted the initial events of trace σ_{wb} . The low-level events were caused by the execution of several high-level activities. The two last columns show which instance of a high-level activity, according to domain expert knowledge, caused the recording of the corresponding low-level events in the same row.

For example, during one execution of the high-level activity *Shift*, i. e., a shift change between nurses, three low-level events are recorded: *NurseChanged* (NC), *CallSignal1* (CS1), and *CallSignal0* (CS0). First, the name of the new responsible nurse is changed in the whiteboard system and recorded as event e_{12} (NC). Then, the call signal system of the hospital recorded event CS1: The nurse entered the room of the patient and pressed a button indicating her presence. After 2 minutes the nurse left the room and pressed another button, which is recorded as CS1 event. All three low-level activities occurred within 2 minutes. The sequence of three events strongly indicates that the high-level activity *Shift* has been executed.

An hour later the call signal system recorded the execution of the low-level activity *CallSignal4* (CS4) as event e_{20} and, again, low-level activity CS1 as event e_{21} . Event CS4 indicates that an alarm button has been pressed and event CS1 indicates, again, that the nurse indicated her presence in the room of the patient. Thus, both events together give evidence that the high-level activity *Alarm* occurred. However, the event CS0, which indicates that the nurse left the room, is *missing*. Maybe the event was not recorded, or the nurse forgot to press the respective button. We denote such missing events as *noise*. Moreover, both high-level activities *Shift* and *Alarm* are known to cause events for the low-level activity CS1 to be recorded. Low-level activity CS1 is an example of *shared functionality*, i. e., the respective button is used in both high-level activities.

Finally, two high-level activities are executed: again an *Alarm* activity instance and, additionally, a *Handover* activity instance. The handover takes place during the alarm, a second nurse *Nathan* takes over the responsibility for the patient, e. g., because nurse *Nancy* is required elsewhere. For the activity *Alarm* the three events CS4, CS1, and CS0 are registered. Moreover, the name of the new nurse *Nathan* is recorded. For the activity *Handover*, only one event recording the execution of low-level activity NC is present. Again, this low-level activity is an example of *shared functionality* since it was also recorded by an instance of the high-level activity *Shift* (cf., e_{32} and e_{12}). Moreover, both high-level activity instance are examples for the *parallel execution* of high-level activities.

9.2 Guided Process Discovery Method

In this chapter, we describe the *Guided Process Discovery method* (GPD). We use event abstraction based on domain knowledge on the process to guide process discovery techniques towards a process model that can be recognized by process stakeholders. We assume that multiple low-level events grouped together indicate the execution of a high-level activity. Moreover, we assume that domain knowledge on the presumed grouping between high-level activities instances and low-level events can be provided. For example, in Table 9.2 the execution of the high-level activity *Shift* is manifested as sequence of three low-level events *NurseChanged*, *CallSignal1*, and *CallSignal0*.

Domain knowledge about the behavior of activities at a higher level of abstraction is captured by *activity patterns*. Activity patterns describe complex interactions of the control-flow, time, resource and, data perspective in terms of low-level events. An activity pattern can be seen as description of the work practice on a fine level of granularity, i.e., low-level of abstraction. Multiple activities on a lower level of abstraction are captured within one activity pattern. With a set of activity patterns at hand, we leverage *alignment techniques* (e.g., the method described in Section 5.2) to find an optimal mapping between the behavior defined by these activity patterns and the observed low-level events in the event log.

In Section 9.2.1, we give an overview of the method and the inputs required. Sections 9.2.2 to 9.2.8 describe the individual steps of the GPD method. In Section 9.3, sketch the implementation of the method. we In Section 9.4, we sketch the evaluation of the method that is based on its application in four case studies. Finally, in Section 9.5 we discuss related work and conclude this chapter in Section 9.6.

9.2.1 Overview of the GPD Method

The input to the GPD method is:

- (A) a **low-level event log** $L_L = (E_L, \Sigma_L, \#^L, \mathcal{E}_L)$ with low-level events E_L describing the execution of a process in terms of low-level activities Σ_L and
- (B) **domain knowledge** on how the recorded low-level activities relate to instances of meaningful high-level activities Σ_H that helps to identify *activity patterns*.

We transform the low-level event log to an *abstracted* event log $L_H = (E_H, \Sigma_H, \#^H, \mathcal{E}_H)$ at the *desired level of abstraction* by using domain knowledge on the relation between low-level activities and high-level activities. Our method can deal with noise, reoccurring and concurrent behavior, and shared functionality.

The GPD method consists of the following 7 steps that are depicted in Figure 9.2:

1. We *identify* and *encode* a set of *activity patterns* that describe domain knowledge on high-level activity behavior. Each activity pattern represent the

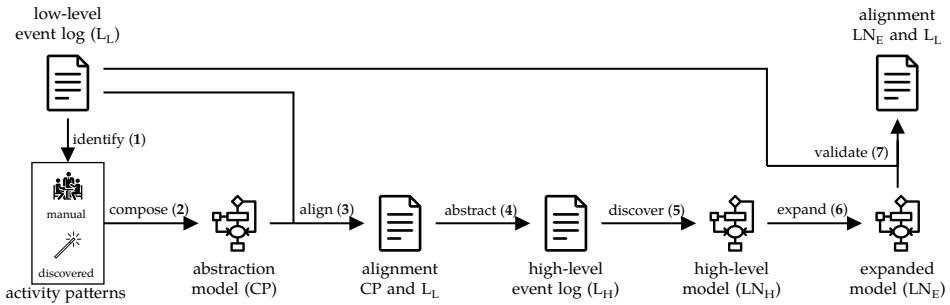


Figure 9.2: Overview of the seven steps of the proposed GPD method.

assumed behavior for one high-level activity.

2. We *compose* activity patterns in an integrated *abstraction model* CP.
3. We *align* the abstraction model CP with the low-level event log L_L . The use of alignments is justified by the fact that, in general, event logs are noisy. Hence, not all low-level events can be mapped onto instance of high-level activities. Furthermore, the search for an optimal mapping requires considering entire traces. Alignment-based techniques do this by solving optimization problems.
4. We *abstract* the low-level event log L_L to a high-level event log L_H using the alignment mapping.
5. Based on this abstracted event log, we can analyze the process at the desired level of abstraction. Often, we want to *discover* a process model LN_H at the desired level of abstraction based on the abstracted high-level event log.

In case a high-level process model LN_H was obtained, we need to validate its quality. The GPD method encompasses two additional steps that can be employed.

6. We *expand* high-level activities in the discovered high-level process model with their corresponding activity patterns to get an expanded process model LN_E .
7. Finally, we *validate* the expanded process model LN_E against the original event log L_L by computing an alignment. This is possible since the expanded process model is defined over the original low-level activities recorded in L_L .

In the following sections, we describe each step of the GPD method in detail.

9.2.2 Encoding the High-level Behavior in Activity Patterns

We represent knowledge about the relation between *low-level activities* Σ_L and *high-level activities* Σ_H with multi-perspective *activity patterns*. An activity pattern can be seen as description of the high-level activity in terms of work practice on a fine level of granularity, i.e., the steps required to execute the high-level activity on a low-level of abstraction. We encode activity patterns as trace sets to allow the

specification of complex interactions of the control-flow, time, resource and, data perspective in terms of low-level events. The trace set specifies those events that are expected to be seen in the event log for *one instance of the corresponding high-level activity*.

Activities have *life-cycles* [Aal16]. Therefore, we introduce a set of life-cycle transitions LT. The LT may include life-cycle transitions such as: schedule, assign, start, suspend, resume, and complete. In the remainder of this chapter, we use only the start and the complete life-cycle transitions, i. e., we assume that $LT = \{\text{start}, \text{complete}\}$.

Definition 9.1 (Activity Pattern). Let U be an universe of values. Let $V_P \subseteq V$ be a set of variables with domain function dom . Let T be the set of transitions used in the activity pattern. Let $\Sigma_H \subseteq U$ be a set of high-level activities. Let LT be a set of life-cycle transition. We define an activity pattern as $ap = (TS, \lambda, v, hl, lt)$:

- (TS, λ, v) , is a labeled trace set that defines the behavior expected for the execution of *one instance* of a high-level activity, i. e., $TS \subseteq (T \times \mathcal{U}_{\text{dom}}^P)^*$ = PS* is the set of execution instances of the pattern.
- $hl : T \rightarrow \Sigma_H$, is a mapping between transitions and high-level activities.
- $lt : T \rightarrow LT$, is a mapping between transitions and life-cycle transitions.

We denote the set of all activity patterns (i. e., atomic and composite) with AP. ◇

We distinguish between *atomic* and *composite* activity pattern. *Atomic activity patterns* encode the behavior of a *single high-level activity instance*, i. e., mapping function hl maps to only one high-level activity: $\forall_{t \in T} \forall_{t' \in T} (hl(t) = hl(t'))$. In each execution trace $\sigma_{ap} \in TS$ of an atomic activity pattern, steps $(t, w) \in \sigma_{ap}$ correspond to the low-level transitions t and the low-level variable assignments w that are expected to be executed as part of the high-level activity $hl(t) \in \Sigma_H$. For composite activity patterns, we do not restrict the range of the mapping function hl , i. e. multiple high-level activities may be encoded. Later in Definition 9.2, we compose several atomic activity patterns to composite activity patterns. Mapping lt is specified by the user and it is motivated by the observation that activities rarely happen instantaneously. Usually, there will be at least one transition being mapped to the *start* life-cycle transition and one transition to the *complete* life-cycle transition.

We require that process transitions are not shared between activity patterns, i. e.:

$$\begin{aligned} & \forall_{(TS_1, \lambda_1, v_1, hl_1, lt_1) \in AP} \\ & \forall_{(TS_2, \lambda_2, v_2, hl_2, lt_2) \in AP} (\{t \in T \mid \exists_{\sigma_{ap} \in TS_1} \exists_{w \in \mathcal{U}_{\text{dom}}^P} ((t, w) \in \sigma_{ap})\} \\ & \quad \cap \{t \in T \mid \exists_{\sigma_{ap} \in TS_2} \exists_{w \in \mathcal{U}_{\text{dom}}^P} ((t, w) \in \sigma_{ap})\} \neq \emptyset) \\ & \implies ((TS_1, \lambda_1, v_1, hl_1, lt_1) = (TS_2, \lambda_2, v_2, hl_2, lt_2)). \end{aligned}$$

Similarly, we require that variables are not shared between activity patterns, so that we can uniquely identify to which pattern a process step belongs. This is

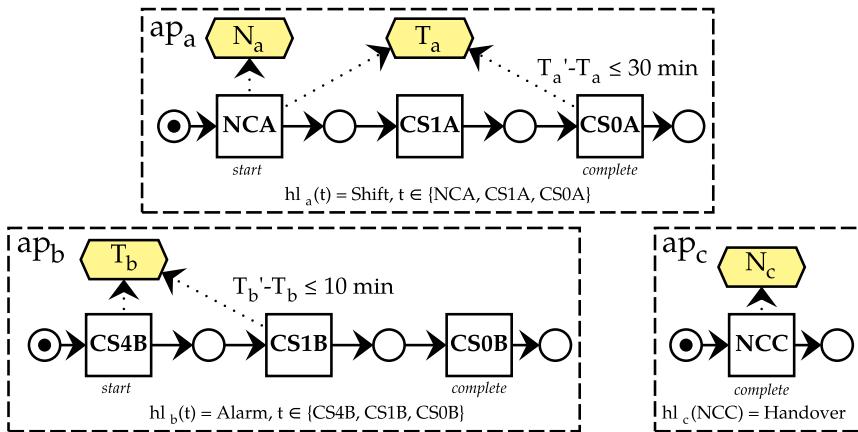


Figure 9.3: Three activity patterns $ap_a, ap_b, ap_c \in AP$ for the example with process models in DPN notation. The respective label functions λ are implicitly encoded in the abbreviated transition names (e.g., $\lambda_a(CS1A) = \text{CallSignal1}$).

not limiting: If this condition does not hold, the process transitions and process variables of the activity pattern can be renamed to avoid overlaps in names. Still, process transitions $t_1, t_2 \in T$ from two different patterns may be associated with the same activity name, i.e., $\lambda_1(t_1) = \lambda_2(t_2)$. Activity patterns can share functionality.

Example 9.2 (Three activity patterns for the hospital whiteboard process). Here, we use DPNs [Man+16c] as language that operationalizes the language-independent GPD method. This is just for the purpose of illustration: Other languages could be used. Figure 9.3 shows three activity patterns $ap_a = (TS_a, \lambda_a, v_a, hl_a, lt_a)$, $ap_b = (TS_b, \lambda_b, v_b, hl_b, lt_b)$ and $ap_c = (TS_c, \lambda_c, v_c, hl_c, lt_c)$ that are implemented as DPNs. We use the abbreviated low-level activity names concatenated with the pattern name for transitions. For example, transition $CS1A$ models activity CallSignal1 , i.e., $\lambda_a(CS1A) = \text{CallSignal1}$. We depict the life-cycle transition mapped to a transition in italics below the transition, e.g., $lt_a(NCA) = \text{start}$.

The **first pattern** ap_a describes the high-level activity Shift , i.e., for all transitions t in the pattern we assign $hl_a(t) = \text{Shift}$. First, the nurse responsible for the patient changes (NCA) and the name of the nurse is recorded in variable N_a . Variable N_a is mapped to the attribute $nurse$, i.e., $v_a(N_a) = \text{nurse}$. Within 30 minutes ($T'_a - T_a \leq 30\text{min}$ with $v_a(T_a) = \text{time}$), the responsible nurse visits the patient and the call signal system records a button press ($CS1A$). Finally, the nurse leaves the room and another button press is registered ($CS0A$) resetting the status.

The **second pattern** ap_b describes a similar sequence (i.e., transitions $CS1B$ and $CS0B$), but represents a different high-level activity: The patient is attended

outside of the normal routine. Transition CS4 has to be executed at most 10 minutes beforehand (i.e., $T'_b - T_b \leq 10\text{min}$). The low-level activity corresponding to CS4B is an *alarm* triggered by the patient. We assign $hl_b(t) = \text{Alarm}$ for each transition t in the pattern.

The **third pattern** ap_c describes a simple handover between nurses: Only the responsible nurse changes (NCC) without any consultation of the patient. We assign $hl_c(t) = \text{Handover}$ for each transition t in the pattern. Transition NCC is an example of shared functionality. Both transitions NCC and NCA are labeled with the same activity name, i.e., $\lambda_a(\text{NCA}) = \lambda_c(\text{NCC}) = \text{NurseChanged}$.

9.2.3 Identifying the Activity Patterns

Activity patterns represent the knowledge about how high-level activities are reflected by low-level events in the event log. Please note that we do not expect an activity pattern to be an *exact representation* of every possible way a high-level activity manifests itself in the event log. In fact, in Section 9.2.5 we show that our method is able to deal with *approximate matches*. Since obtaining suitable activity patterns is crucial for the GPD method, we elaborate here on how to obtain them. We categorize activity patterns based on the way they have been obtained into: manual patterns and discovered patterns. Table 9.3 provides a list of sources and examples for activity patterns.

Table 9.3: Sources for manual and discovered activity patterns.

Source	Category	Examples
Expert knowledge	Manual	Patterns ap_a , ap_b , ap_c (Figure 9.3)
Process questions	Manual	Different admission variants (Chapter 13)
Standard models	Manual	Transactional life-cycle model [Aal16] (Figure 9.4), clinical protocols
Local behavior	Discovered	Local process models [Tax+16a], frequent sub sequences [JA09], episodes [LA15], instance graphs [DGP16]
Decomposed behavior	Discovered	Region theory [Car12], clustering [HVA14]
Data attributes	Discovered	Discovery per department (Chapter 13)

Manual patterns

Manual patterns are created based on domain knowledge about the high-level activities of the process at hand. We further subdivided manual patterns based

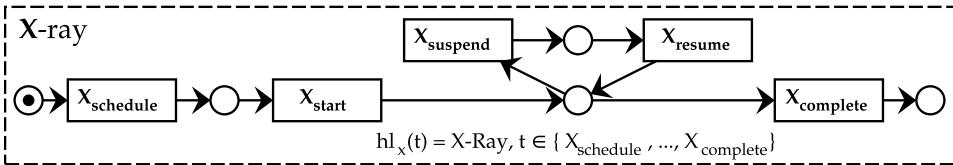


Figure 9.4: An activity pattern capturing the life-cycle of the high-level activity X-Ray.

on the source of the employed domain knowledge into patterns based on *expert knowledge*, *process questions*, and *standard models*.

Expert knowledge. Manual patterns based on *expert knowledge* encode assumptions on the system. Stakeholders of the process can provide initial assumptions on how high-level activities are manifested in the event log. Moreover, semantically related activities can be grouped together to form sub-processes. If the expected behavior of such a sub-process is known, it can be captured as an activity pattern. The subprocess captured by the activity pattern can be seen as a single activity on a higher level of abstraction. In Example 9.2 (Figure 9.3), we describe three activity patterns that are based on expert knowledge. The patterns encode the assumption that the low-level activities CS1 and CS0 both occur in the context of shift change and in the context of an alarm. This knowledge was obtained from a domain expert who is familiar with the system.

Process questions. Often, questions on the process can be used as a source for activity patterns. These patterns are not driven by knowledge about the questions, but knowledge about the required type of output. For example, in one of the case studies (cf., Chapter 13) we use an activity pattern that is based on the process question: "What are the trajectories of patients in the hospital based on their admission?". The activity pattern encodes the different variants of the admission.

Standard models. Some behavioral patterns appear across several domains. Often, these are based on domain independent standard models, e. g., the transactional life-cycle model [Aal16]. Discovering such patterns from event logs is challenging for state-of-the-art process discovery algorithms. For example, a specialized algorithm exist for event logs with life-cycle information [LFA16]. It is possible to encode the expected behavior as activity patterns and, thus, to use standard algorithms. For example, in Figure 9.4 we show how to adapt the transactional life-cycle model to encode the transitions for the high-level activity X-Ray. An X-Ray is scheduled ($x_{\text{scheduled}}$), started (x_{start}), possibly suspended ($x_{\text{suspended}}$), resumed (x_{resumed}), and eventually completed (x_{complete}). The XES standard [IEECCIS16] defines an extension for the transactional life-cycle model.

Discovered patterns

It is also possible to automatically discover patterns from the low-level event log. We distinguish between patterns that are discovered based on *local behavior*, based on *decomposed behavior*, and based on *data attributes*.

Local behavior. There are dedicated pattern mining techniques [DGP16; JA09; LA15; Tax+16a] that discover patterns of local behavior from event logs. Such patterns do not capture the behavior of the complete traces. The models describe subsets of the events and the same event can be part of several patterns. Such discovered local patterns can be directly used as input to the GPD method, e.g., in [MT17] we sketched an initial exploration of using local process models as patterns. However, it can be challenging to automatically assign good labels to discovered patterns and there is no guarantee that local process models, indeed, represent abstract activities. Methods for automatic labeling of process fragments [Leo+14b] do exist.

Decomposed behavior. Work on decomposed process discovery [Aal13; Car12; HVA14] could be leveraged to obtain activity patterns that represent parts of the observed behavior. Different from methods that discover patterns of local behavior, the event log is decomposed into several disjunct sub-logs using an automated technique. Then, a full process model is discovered for each of the logs using standard process discovery techniques. This process model can be used as activity pattern.

Data attributes. Next to automatic decomposition approaches, information can be exploited on the hierarchical structure that is stored in the data attributes of the event log. For example, later in the evaluation (Section 9.4), we use information on the department in which an event occurred. We split the event log into sub logs based on the department and discover three separate process models that are used as activity patterns.

9.2.4 Composing the Activity Patterns to an Abstraction Model

With a set of activity patterns for the process under analysis at hand, we compose their behavior into an integrated *abstraction model* by using composition functions.

Composition Functions

The composition of activity patterns may restrict the interaction that is possible between the high-level activities that are captured by the activity patterns. By restricting the interaction, we help to find a more precise mapping between low-level events and high-level activities.

Definition 9.2 (Composition Function). A composition function $f : AP^* \rightarrow AP$ combines activity patterns ap_1, \dots, ap_n into a composite activity pattern $cp \in AP$, i.e., $f(ap_1, \dots, ap_n) = cp$. We denote the set of all composition functions as $F : AP^* \rightarrow AP$. \diamond

Given activity patterns $ap_i = (TS_i, \lambda_i, v_i, hl_i, lt_i) \in AP$ with $i \in \mathbb{N}$ and $TS_i \subseteq (T \times \mathcal{U}_{dom}^P)^*$ $= PS_i^*$, we define the semantics for five basic composition functions: *sequence*, *choice*, *parallel*, *interleaving* and *repetition*. Our abstraction method is not restricted to these functions. Further composition functions can be added.

- **Sequence** composition $\odot \in F$ with $dom(\odot) = \{(ap_1, ap_2)\}$:

$$\begin{aligned} ap_1 \odot ap_2 &= (TS, \lambda, v, hl, lt) \text{ with} \\ TS &= \{\sigma_1 \cdot \sigma_2 \mid \sigma_1 \in TS_1 \wedge \sigma_2 \in TS_2\}, \text{ and} \\ \lambda &= \lambda_1 \oplus \lambda_2, v = v_1 \oplus v_2, hl = hl_1 \oplus hl_2, lt = lt_1 \oplus lt_2. \end{aligned}$$

The binary operation \odot is associative. We write $\odot_{1 \leq i \leq n} ap_i = ap_1 \odot ap_2 \odot \dots \odot ap_n$ to compose sequences of patterns in sequence. Moreover, we define $\odot_{1 \leq i \leq 0} ap_i = \{\langle \rangle\}$.

- **Choice** composition $\otimes \in F$ with $dom(\otimes) = \{(ap_1, ap_2)\}$:

$$\begin{aligned} ap_1 \otimes ap_2 &= (TS, \lambda, v, hl, lt) \text{ with} \\ TS &= TS_1 \cup TS_2, \text{ and} \\ \lambda &= \lambda_1 \oplus \lambda_2, v = v_1 \oplus v_2, hl = hl_1 \oplus hl_2, lt = lt_1 \oplus lt_2. \end{aligned}$$

The binary operation \otimes is commutative and associative. We write $\otimes_{1 \leq i \leq n} ap_i = ap_1 \otimes ap_2 \otimes \dots \otimes ap_n$ to compose sequences of patterns in choice.

- **Parallel** composition $\diamond \in F$ with $dom(\diamond) = \{(ap_1, ap_2)\}$:

$$\begin{aligned} ap_1 \diamond ap_2 &= (p, \lambda, v, hl, lt) \text{ with} \\ TS &= \{\sigma \in (PS_1 \cup PS_2)^* \mid proj(\sigma, PS_1) \in TS_1 \wedge proj(\sigma, PS_2) \in PS_2\} \text{ and} \\ \lambda &= \lambda_1 \oplus \lambda_2, v = v_1 \oplus v_2, hl = hl_1 \oplus hl_2, lt = lt_1 \oplus lt_2. \end{aligned}$$

The binary operation \diamond is commutative and associative. We write $\diamond_{1 \leq i \leq n} ap_i = ap_1 \diamond ap_2 \diamond \dots \diamond ap_n$ to compose sequences of patterns in parallel.

- **Interleaving** composition $\leftrightarrow \in F$ with $dom(\leftrightarrow) = AP^*$ and $p(n)$ denoting the set of all permutations of the numbers $\{1, \dots, n\}$:

$$\leftrightarrow (ap_1, \dots, ap_n) = \bigotimes_{(i_1, \dots, i_n) \in p(n)} \bigodot_{1 \leq k \leq n} ap_{i_k}.$$

- **Repetition** composition $[n, m] \in F$ with $\text{dom}([n, m]) = AP$, $n \in \mathbb{N}_0$, $m \in \mathbb{N} \cup \{\infty\}$, and $n \leq m$:

$$\text{ap}_1^{[n,m]} = \bigotimes_{n \leq i \leq m} \bigodot_{1 \leq k \leq i} \text{ap}_1.$$

We build a composed abstraction model $cp = (TS, \lambda, \nu, hl, lt) \in AP$ with a formula that composes all patterns of interest. The trace set $TS \subseteq PS^*$ corresponds to the overall behavior that we expect to observe for the execution of *all* high-level activities in a single process instance.

Example 9.3 (Composition of activity patterns). Given the activity patterns AP_a , AP_b and AP_c shown in Figure 9.3, we can compose their behavior to

$$cp = (\leftrightarrow (\text{ap}_a^{[0,\infty]}, \text{ap}_b^{[0,\infty]}))^{[0,\infty]} \diamond \text{ap}_c^{[0,\infty]}.$$

Here, we allow that all of the three patterns are repeated an unbounded number of times and may be skipped, i. e., the repetition composition $[0, \infty]$. We allow the absence of patterns using the repetition composition as the corresponding high-level activities might not have been executed in every process instance. Then, we first interleave patterns $\text{ap}_a^{[0,\infty]}$ and $\text{ap}_b^{[0,\infty]}$, i. e., instances of the high-level activity represented by ap_a need to finish before instances of the high-level activity represented by ap_b can start and vice versa. The combined behavior may, again, be repeated indefinitely and may also be skipped. We restrict cp to only contain the interleaving of patterns ap_a and ap_b as there is only one responsible nurse per patient. Therefore, the activities expressed by ap_a and ap_b can occur in any order but should not happen in parallel. Finally, we compose the combined behavior in parallel with pattern $\text{ap}_c^{[0,\infty]}$, i. e., the high-level activity represented by ap_c may be executed in parallel to both ap_a and ap_b . We add ap_c using the parallel composition as handovers can take place in parallel to ap_a and ap_b .

The result of this composition is the abstraction model cp . Model cp corresponds to all behavior that could be observed for executions of the three high-level activities. For example, process trace

$$\langle (NCA, (N_a \mapsto \text{Nancy}, T_a \mapsto 0)), (CS1A, \emptyset), (NCC, (N_c \mapsto \text{Nathan})), \\ (CS0A, (T_a \mapsto 29)) \rangle$$

belongs to the set of process traces of the composed abstraction model. Whereas the process trace

$$\langle (NCA, (N_a \mapsto \text{Nancy}, T_a \mapsto 0)), (CS1A, \emptyset), (CS4B, \emptyset), (CS0A, (T_a \mapsto 29)) \rangle$$

is not part of the process behavior of cp .

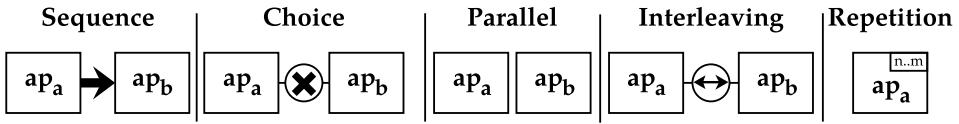


Figure 9.5: Overview of the graphical notation for the composition functions.

We designed a graphical representation for each composition function, which can be used to design abstraction models in the implementation of our approach. It is also possible to use BPMN, Petri nets, or Process trees to visualize the composition of patterns. However, abstraction models are not meant to be full specifications of a process. Mostly basic composition functions, such as parallel, interleaving, and repetition are used. Process discovery based on the resulting high-level event log is still necessary. Therefore, we use this compact graphical representation.

Figure 9.5 shows the graphical notation for each of the introduced composition functions: sequence, choice, parallel, interleaving, and repetition. We attach the unary repetition composition directly to the patterns. If necessary, we draw a box around composed patterns to clarify the precedence of operations. Patterns composed in parallel to each other do not influence the execution of each other. Therefore, we do not draw any edge between the patterns. The interleaving composition is depicted by connecting interleaved patterns to the interleaving operator \leftrightarrow . Patterns are composed in choice by connecting all patterns in choice to a choice operator \otimes . The sequence composition is depicted by a directed edge between two patterns.

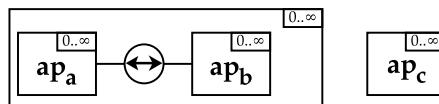


Figure 9.6: Abstraction model cp created by composing the patterns ap_a , ap_b , and ap_c .

Example 9.4 (Abstraction model for the whiteboard process). For example, Figure 9.6 shows the graphical representation of the composition of activity patterns ap_a , ap_b , and ap_c to the abstraction model

$$cp = (\leftrightarrow (ap_a^{[0,\infty]}, ap_b^{[0,\infty]}))^{[0,\infty]} \diamond ap_c^{[0,\infty]},$$

which was introduced in Example 9.3.

Implementation of Composition Functions as DPN

Each of the proposed composition functions is implemented using the DPN notation as described in Figure 9.7. We assume that activity patterns $ap_a = (p_a, \lambda_a, v_a, hl_a, lt_a)$, $ap_b = (p_b, \lambda_b, v_b, hl_b, lt_b) \in P$ are implemented as DPNs with source places s_a, s_b and sink places e_a, e_b .²⁸ We describe how to compose ap_a and ap_b to a combined pattern for each of the introduced composition functions. We focus on the composition of their process models p_a and p_b since the remaining mapping functions are combined by taking their union.

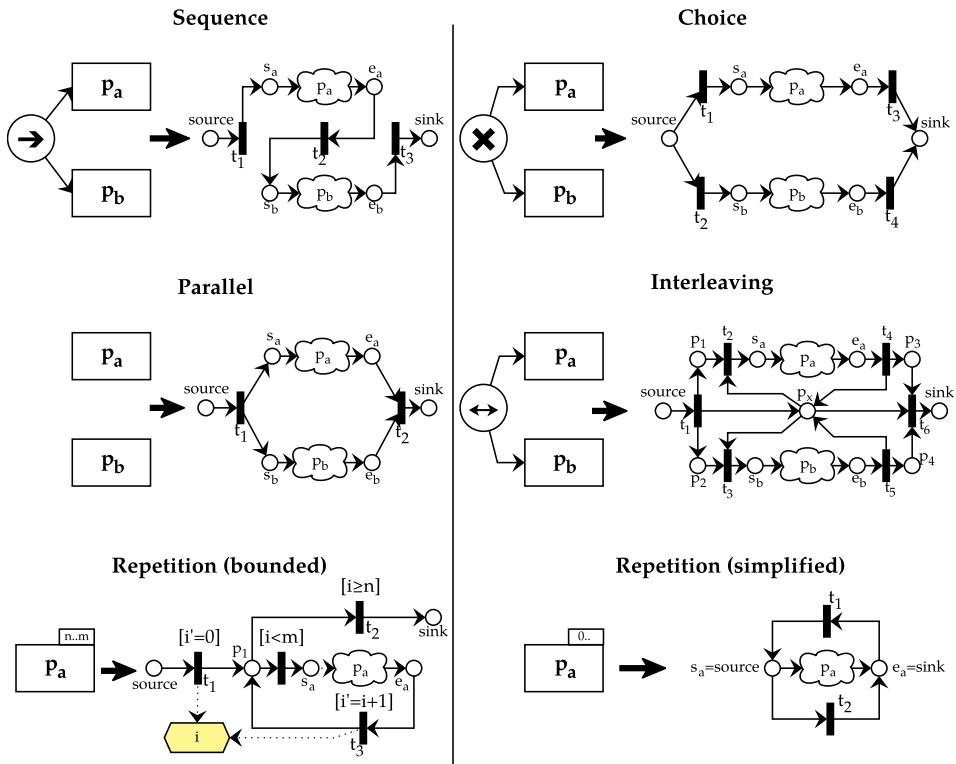


Figure 9.7: Implementation of the composition functions using the DPN notation

Sequence. Pattern ap_a and pattern ap_b are composed in sequence by adding two places (source, sink) as the entry and exit points of the composed pattern. Moreover,

²⁸Requiring single source and single sink places is not a limitation. Any Petri net can be transformed in this way by adding invisible transitions.

transitions t_1, t_2, t_3 are added to connect the source places s_a, s_b and sink places e_a, e_b of both patterns in sequence.

Choice. Pattern ap_a and pattern ap_b are composed in choice by adding two places (*source, sink*) as the entry and exit points of the composed pattern. Additionally four transitions (t_1, t_2, t_3, t_4) are added. The control-flow is split after place *source* such that either t_1 or t_2 has to be executed. Transition t_1 is connected to the source place s_a of pattern ap_a and transition t_2 is connected to the source place s_b of pattern ap_b . The sink places of the patterns are connected the exit place *sink* via transitions t_3 and t_4 , respectively.

Parallel. Pattern ap_a and pattern ap_b are composed in parallel by adding two places (*source, sink*) and two transitions (t_1, t_2). The control-flow is split using transition t_1 such that both patterns p_a and p_b have to be executed. The exit places e_a and e_b are connected to transition t_2 , which merges the parallel branches.

Interleaving. Pattern ap_a and pattern ap_b are composed in interleaving by adding seven places (*source, sink, p₁, p₂, p₃, p₄* and p_x) and six transitions ($t_1, t_2, t_3, t_4, t_5, t_6$) as shown in Figure 9.7. Intuitively, the interleaving of p_a and p_b can be expressed as choice between any possible ordering of p_a and p_b . The control-flow is split in parallel using t_1 enabling any possible re-ordering of patterns p_a and p_b . Place p_x is added restricting the behavior such that only either p_a or p_b can be executed at the same time. Finally, transition t_6 merges the control-flow from places p_x, p_3 and p_4 .

Repetition. The repetition of a pattern ap_a is modeled by adding three places (*source, sink, p₁*) and three transitions (t_1, t_2, t_3). We use a counter variable i that keeps track of the repetitions and add guards to transitions t_1, t_2 and t_3 that constrain the maximum allowed and minimum required number of repetitions accordingly. Transition t_3 increases the counter i on each iteration. Because we have a-priori knowledge of the number of repetitions such a construct can always be unfolded to a normal Petri net, e. g., by repeated use of the sequence and choice composition and duplicating the pattern. In the case $m = \infty$, the guard $i < m$ can be removed. Moreover, in case the number of repetitions is unbounded, i.e., $m = \infty$ and $n = 0$ we can simplify the construction as shown on the right-hand side of Figure 9.7.

Example 9.5 (Implementation of an abstraction model with a DPN). Figure 9.8 depicts the DPN implementation of the abstraction model cp . To simplify the composition, we assume that the DPNs of activity patterns have a single source place and a single sink place. The abstraction model starts with a single source place *src* and ends with a single sink place *snk*. We model the parallel

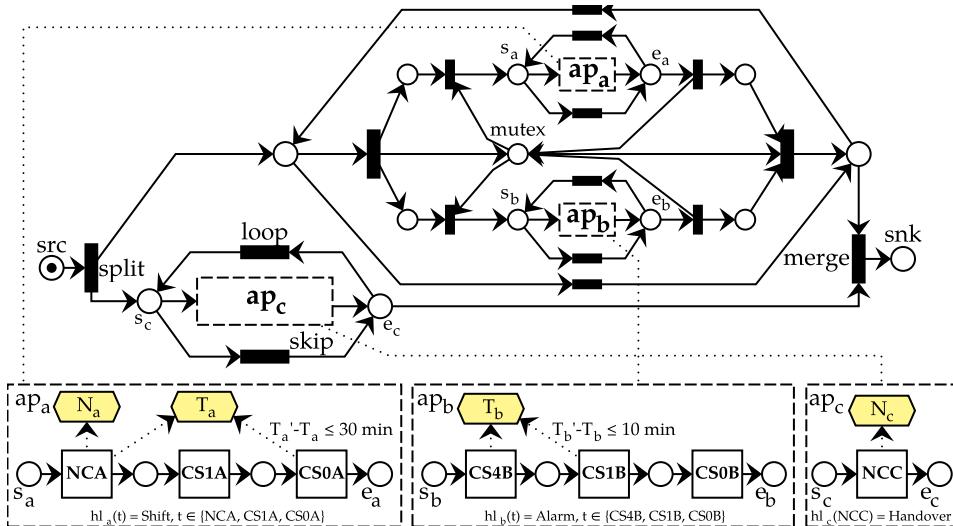


Figure 9.8: DPN created by our implementation for the abstraction model cp. The process models of the activity patterns ap_a , ap_b , ap_c are depicted as dashed rectangles P_a , P_b , P_c with source places s_a , s_b , s_c and sink places e_a , e_b , e_c . Black transitions are invisible routing transitions, which are not recorded in event logs.

composition of $ap_c^{[0,\infty]}$, i. e.:

$$\leftrightarrow (ap_a^{[0,\infty]}, ap_b^{[0,\infty]})^{[0,\infty]}$$

by adding invisible transitions *split* and *merge*, which realize a parallel split and join. Invisible transitions cannot be observed; they are only added for routing purposes. We use place *mutex* to model the mutual exclusion constraint of the interleaving composition of patterns $ap_a^{[0,\infty]}$ and $ap_b^{[0,\infty]}$. Place *mutex* guarantees that only either ap_a or ap_b can be executed at the same time, yielding the interleaving of ap_a and ap_b . Each repetition composition is implemented by adding two invisible transitions *loop* and *skip*, which allow to repeat the pattern indefinitely or to skip its execution, respectively.

9.2.5 Aligning the Event Log and the Abstraction Model

With an abstraction model at hand, we need to relate the behavior in the low-level event log to process traces defined by the abstraction model $cp = (TS, \lambda, \nu, hl, lt)$. More specifically, we need to determine the mapping between low-level events in traces $\sigma \in \mathcal{E}_L$ of the event log and process steps in process traces of the labeled trace

set $\text{LTS} = (\text{TS}, \lambda, \nu)$ that is defined by the abstraction model cp . Concretely, we use the *alignment* technique for DPNs presented in Chapter 5 to establish alignments $\gamma_\sigma \in \Gamma_{\text{L}_L, \text{LTS}}^\sigma$ between log traces and process traces. The alignment guarantees that its sequence of model steps without \gg -steps is a process trace defined by the composed abstraction model. Moreover, by computing an optimal alignment, we guarantee that we obtain one of the process traces with the least deviations from the events recorded in the log trace σ . Please note that we can uniquely identify sub-sequences of the initial (uncomposed) activity patterns in the alignment since we required transitions to be unique among activity patterns.

Table 9.4: The first two columns show an excerpt of an alignment $\gamma_{\sigma_{\text{wb}}} \in \Gamma_{\text{L}_L, \text{LTS}}^{\sigma_{\text{wb}}}$ between the whiteboard example log trace $\sigma_{\text{wb}} \in \mathcal{E}_L$ and the trace set of the composed abstraction model LTS. Low-level events $e \in \mathcal{E}_L$ are paired with process steps (t, w) . The first row shows the attributes $\#(e)$ of the respective aligned events. The last two columns depict an excerpt of an abstracted trace $\sigma_H \in \mathcal{E}_H$ from created high-level event log. The events $\hat{e} \in \mathcal{E}_H$ and corresponding attributes $\#^H(\hat{e})$ are returned by the GPD method.

alignment $\gamma_{\sigma_{\text{wb}}}$		high-level trace σ_H	
$\#(e)$	$(e, (t, w))$	\hat{e}	$\#^H(\hat{e})$
(activity \mapsto NC, time \mapsto 122, nurse \mapsto Nancy)	● (e ₁₂ , (NCA, w ₁))	\hat{e}_5	(activity \mapsto Shift, time \mapsto 122, nurse \mapsto Nancy, lifecycle \mapsto start, instance \mapsto 3)
(activity \mapsto CS1, time \mapsto 122)	● (e ₁₃ , (CS1A, w ₂))	-	-
(activity \mapsto CS0, time \mapsto 124)	● (e ₁₄ , (CS0A, w ₃))	\hat{e}_6	(activity \mapsto Shift, time \mapsto 124, lifecycle \mapsto complete, instance \mapsto 3)
...
(activity \mapsto CS4, time \mapsto 185)	● (e ₂₀ , (CS4B, w ₄))	\hat{e}_{11}	(activity \mapsto Alarm, time \mapsto 185, lifecycle \mapsto start, instance \mapsto 6)
(activity \mapsto CS1, time \mapsto 194)	● (e ₂₁ , (CS1B, w ₅))	-	-
-	● (gg, (CS0B, w ₆))	\hat{e}_{12}	(activity \mapsto Alarm, time \mapsto 194, lifecycle \mapsto complete, instance \mapsto 6)
...
(activity \mapsto CS4, time \mapsto 310)	● (e ₃₀ , (CS4B, w ₇))	\hat{e}_{21}	(activity \mapsto Alarm, time \mapsto 310, lifecycle \mapsto start, instance \mapsto 11)
(activity \mapsto CS1, time \mapsto 311)	● (e ₃₁ , (CS1B, w ₈))	-	-
(activity \mapsto NC, time \mapsto 311, nurse \mapsto Nathan)	● (e ₃₂ , (NCC, w ₉))	\hat{e}_{22}	(activity \mapsto Handover, time \mapsto 312, nurse \mapsto Nathan, lifecycle \mapsto complete, instance \mapsto 12)
(activity \mapsto CS0, time \mapsto 315)	● (e ₃₃ , (CS0B, w ₁₀))	\hat{e}_{23}	(activity \mapsto Alarm, time \mapsto 315, lifecycle \mapsto complete, instance \mapsto 11)

Example 9.6 (Alignment between low-level log and composed abstraction model). The first two columns of Table 9.4 show an excerpt of an alignment $\gamma_{\sigma_{wb}} \in \Gamma_{L_L, LTS}^{\sigma_{wb}}$ between the whiteboard example log trace (Table 9.2) and a process trace of the abstraction model introduced in Figure 9.6. The first column repeats the attributes of the low-level events and the second column shows the sequence of alignment moves. Thus, the process projection (Section 4.3) of the sequence depicted in the second column is a process trace of the abstraction model. From the executed transitions, it can be deduced that both patterns ap_a and ap_c are executed once and pattern ap_b is executed twice. For example, the sub-sequence $\langle (CS4B, w_4), (CS1B, w_5), (CS0B, w_6) \rangle$ captures one execution of the activity pattern ap_b , i. e., this sub-sequence would be a process trace of activity pattern ap_b . Please note that step $(CS0B, w_6)$ was inserted as model move (\bullet) by the alignment, i. e., there is no corresponding event in the event log. Thus, the execution instance of activity ap_b was only partially observed in the event log. Moreover, the execution instances of two activity patterns may overlap as is the case with the activity pattern ap_c , which consists only of transition NCC. Transition NCC was mapped to event e_{32} , which occurred while an instance of activity pattern ap_b was not yet completed.

9.2.6 Abstracting the Event Log

We describe how to build the high-level event log $(E_H, \Sigma_H, \#^H, \mathcal{E}_H)$ using an alignment of the low-level event log with the abstraction model. In general, there might be scenarios where one event could be mapped to several activity instances. We simplify the discussion by assuming that events are only mapped to single activity instances. This is not a limitation, as described by Baier et al. [BMW14]: Those events could be duplicated in a pre-processing step beforehand.

Abstraction Algorithm

The last two columns of Table 9.4 show how we obtain the high-level event log from the information provided by the alignment. We align each trace $\sigma \in \mathcal{E}_L$ of the low-level event log with the abstraction model $cp = (TS, \lambda, \nu, hl, lt)$ and obtain an alignment $\gamma_\sigma \in \Gamma_{L_L, LTS}^\sigma$, as shown in the second column of Table 9.4. Then, we use this alignment and the mapping functions hl and lt to build the high-level event log as specified by Algorithm 4. New events \hat{e} for high-level activities are added to E_H for those alignment moves $(e, (t, w))$ for which transition t is mapped to a life-cycle transition, i. e., in our case either *start* or *complete*. Those transitions correspond to visible life-cycle transitions in the life-cycle of the high-level activity.

In Algorithm 5, we show how each new event \hat{e} is assigned a name based on the

```

Input: Low-level Event Log  $L_L = (E_L, \Sigma_L, \#^L, \mathcal{E}_L)$ , Abstraction Model  

         $cp = (TS, \lambda, v, hl, lt)$   

Result: High-level Event Log  $L_H = (E_H, \Sigma_H, \#^H, \mathcal{E}_H)$ 

1  $LTS \leftarrow (TS, \lambda, v)$  // labeled trace set of cp
2  $E_H \leftarrow \emptyset, \Sigma_H \leftarrow \{act_H \mid \exists_{act_H \in \Sigma} (hl(act_L) = act_H)\}$ 
3  $ai : \Sigma_H \rightarrow \mathbb{N}_0$  s.t.  $\forall_{act_H \in \Sigma_H} (ai(act_H) = 0)$ 
4 for  $\sigma \in \mathcal{E}_L$  do
5    $\gamma_\sigma \leftarrow optimalAlignment(\sigma, LTS)$  // cf. Definition 4.4
6    $\sigma_H \leftarrow \langle \rangle$ 
7   for  $(e, s) \in \gamma_\sigma$  s.t.  $s \neq \gg$   $\wedge s = (t, w) \wedge t \in dom(lt)$  do
8      $E_H \leftarrow E_H \cup \{\hat{e}\}$ 
9      $\sigma_H \leftarrow \sigma_H \cdot \langle \hat{e} \rangle$ 
10     $assignAttributes(\hat{e}, (e, (t, w)), \gamma_\sigma, ai, hl, lt)$  // activity name, etc.
11    if  $lt(t) = complete$  then  $ai(hl(t)) \leftarrow ai(hl(t)) + 1$  // next instance
12   $\mathcal{E}_H \leftarrow \mathcal{E}_H \cup \{\sigma_H\}$ 
13 return  $(E_H, \Sigma_H, \#^H, \mathcal{E}_H)$ 

```

Algorithm 4: Procedure building a high-level event log based on an abstraction model and a low-level event log.

mapping $hl(t)$, a *unique activity instance identifier*²⁹ for each execution of an activity pattern, and its life-cycle transition obtained from the mapping $lt(t)$. Moreover, we copy the values of all variables defined in the abstraction model $v \in dom(w)$ to the attributes of the new high-level event \hat{e} . In this manner, we create a high-level trace in \mathcal{E}_H for each low-level trace in \mathcal{E}_L . After processing all low-level traces in this manner, Algorithm 4 returns the abstracted high-level log $L_H = (E_H, \Sigma_H, \#^H, \mathcal{E}_H)$.

Example 9.7 (Building of a high-level event log based on Algorithm 4). For example, events \hat{e}_5 and \hat{e}_6 in Table 9.4 are created based on the alignment of low-level events e_{12} and e_{14} to transitions NCA and CS0A. We assign event \hat{e}_5 the activity name Shift, i.e., $\#_{activity}^H(\hat{e}_5) = Shift$. We assign the unique activity instance identifier 3 to both events, i.e., $\#_{instance}^H(\hat{e}_5) = \#_{instance}^H(\hat{e}_6) = 3$. Instance 3 of the high-level activity Shift was started by event \hat{e}_5 and completed by event \hat{e}_6 . Then, we assign the life-cycle transition start to \hat{e}_5 (i.e., $\#_{lifecycle}^H(\hat{e}_5) = start$) and the life-cycle transition complete to event \hat{e}_6 . Finally, we copy the values of the variables N_A and T_A as $\#_{nurse}^H(\hat{e}_5) = Nancy$ and $\#_{time}^H(\hat{e}_5) = 122$.

We ensure that every high-level event $\hat{e} \in E_H$ is assigned a *timestamp*. We consider two cases depending on the alignment move $(e, (t, w))$:

²⁹A unique instance identifier *concept:instance* is defined in the concept extension of the XES standard [IEECCIS16].

```

Input: Event  $\hat{e}$ , Alignment move  $(e, (t, w))$ , Alignment  $\gamma_\sigma$ , Instance mapping  $ai$ ,
High-level activity mapping  $hl$ , Life-cycle mapping  $lt$ 

1  $\#^H(\hat{e})(\text{activity}) \leftarrow hl(t)$  // assign activity name
2  $\#^H(\hat{e})(\text{instance}) \leftarrow ai(hl(t))$  // assign activity instance
3  $\#^H(\hat{e})(\text{lifecycle}) \leftarrow lt(t)$  // assign life-cycle
4 for  $v \in \text{dom}(w)$  do  $\#^H(\hat{e})(v(v)) \leftarrow w(v)$  // copy variables
5 if  $e \neq \gg$  then
6    $\#^H(\hat{e})(\text{time}) \leftarrow \#_{\text{time}}^L(e)$  // assign timestamp
7 else
8    $\#^H(\hat{e})(\text{time}) \leftarrow obtainTime(\gamma_\sigma, (e, (t, w)), lt(t))$ 

```

Algorithm 5: Procedure *assignAttributes* that assign the attributes of newly created events. Each event represents the execution of a transition in the life-cycle of a high-level activity instance.

1. the process step was aligned to a low-level event $e \in E_L$ and
2. the process step was mapped to $e = \gg$, i. e., a model move.

In the first case, we assign the timestamp of the aligned low-level event to the high-level event. For example, $\#_{\text{time}}^H(\hat{e}_{11}) = \#_{\text{time}}(e_{20}) = 185$. In the second case, there are several possibilities to determine the most likely timestamp for a model move (e. g., based on statistical methods [Rog+13]). Therefore, we abstract from the concrete implementation of the method by using function *obtainTime*, which may be plugged-in depending on the use case. For all case studies presented in Chapters 12 to 15, we implemented *obtainTime* by using the timestamps of the closest neighboring low-level (i. e., the earliest event for the *start* life-cycle and the latest event for the *complete* life-cycle) events that are mapped to the same high-level activity instance. For example, for the unmapped high-level event \hat{e}_{12} , we use the timestamp from the neighboring event e_{21} , i. e., $\#_{\text{time}}^H(\hat{e}_{12}) = \#_{\text{time}}(e_{21}) = 194$.

Quality of the Abstraction

The alignment enables the definition of two quality measures for the abstraction mapping. First, we use **global matching error** $\epsilon_{L_L, cp} \in [0, 1]$ as a measure for how well the entire low-level event log L_L matches the behavior imposed by the composed abstraction model cp . In this context, a fitness measure such as the one defined in Section 4.4 can be seen as global measure for the quality of the used abstraction model. A low fitness indicates that there are many events that cannot be correctly matched, which indicates that the abstraction model does not capture the entire process well. The resulting abstracted event log does not allow a reliable analysis when the overall fitness is low. In case of low fitness, the assumptions made in the identification and encoding of the activity patterns should be revised.

Second, we define a **local matching error** $\epsilon_{L_L, cp}(\text{act}_H) \in [0, 1]$ for each recog-

nized high-level activity $\text{act}_H \in \Sigma_H$. Some process steps in the alignment are not matched to an event in the log, i.e., the event is missing. To obtain $\epsilon_{L, cp}(\text{act}_H)$, we determine the fraction of incorrect and model moves for process activities that are part of the activity pattern for the high-level activity act_H over the total number of all alignment moves for act_H . For the local matching error, we do not consider the log moves since we cannot reliably attribute them to a specific high-level activity. However, log moves are accounted for in the fitness score used by the global matching error.

Definition 9.3 (Local matching error). Let $L_L = (E_L, \Sigma_L, \#^L, \mathcal{E}_L)$ be a low-level event log. Let $cp = (TS, \lambda, v, hl, lt) \in AP$ be an abstraction model. Let $LTS = (TS, \lambda, v)$ be the labeled trace set of cp . We define the matching error $\epsilon_{L, cp}(\text{act}_H) \in [0, 1]$ as the fraction of incorrect and model moves over the total number of all moves for process activities that are mapped to high-level activity $\text{act}_H \in \Sigma_H$ in the alignment $\gamma_\sigma \in \Gamma_{L_L, LTS}^\sigma$ between log traces σ and the labeled trace set LTS :

$$\epsilon_{L, cp}(\text{act}_H) = \frac{\sum_{\sigma \in \mathcal{E}} |\text{proj}(\gamma_\sigma, \Gamma^{\text{err}}(\text{act}_H))|}{\sum_{\sigma \in \mathcal{E}} |\text{proj}(\gamma_\sigma, \Gamma(\text{act}_H))|}, \text{ where}$$

$$\Gamma(\text{act}_H) = \{(e, s) \in \Gamma_{L_L, LTS} \mid s \neq \gg \wedge s = (t, w) \wedge hl(t) = \text{act}_H\}, \text{ and}$$

$$\Gamma^{\text{err}}(\text{act}_H) = \{(e, s) \in \Gamma(\text{act}_H) \mid (\gg, s) \text{ is a model move or}$$

$$(e, s) \text{ is an incorrect synchronous move}\}. \quad \diamond$$

The matching error can be used to exclude individual unreliable activity pattern matches, which exceed a certain ϵ -threshold.

Example 9.8 (Local matching error). For example, in Table 9.4 one execution of process activity CS0B in the activity pattern for the Alarm activity is mapped to \gg . The local matching error $\epsilon_{L_{wb}, cp}(\text{Alarm})$ is $\frac{5}{6}$ for high-level activity Alarm based on the alignment shown in Table 9.4.

9.2.7 Discovering a High-Level Process Model

After creating the abstracted event log L_H , we discover a process model based on the abstracted high-level activities Σ_H . For the process discovery, any state-of-the-art process discovery technique can be employed. However, as the abstracted event log contains information on the life-cycle of activities (i.e., the *start*- and the *complete* transition), we propose to use a process discovery technique that can harness this information, such as the Inductive Miner with the life-cycle extension as proposed in [LFA16]. We could use the discovered process model directly for further analysis (e.g., performance analysis, deviation analysis, decision rule mining etc.).

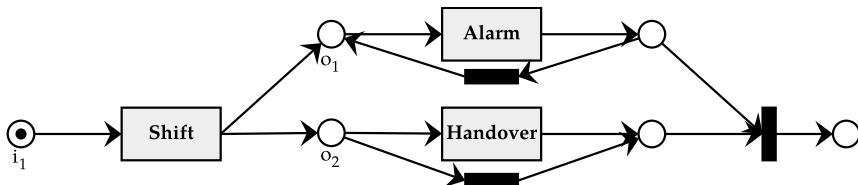


Figure 9.9: A discovered high-level model in DPN notation. We distinguish high-level activities from transitions used in activity patterns by using a gray background.

Example 9.9. Figure 9.9 shows an example of a high-level model in DPN notation that can be discovered based on the abstracted event log. The discovered process starts with a change of shifts. Then, in parallel, patients raise one or more alarms and, sometimes, a handover takes place. In fact, in Table 9.4 the Alarm and the Handover activity are observed to be overlapping each other. An instance of the activity Handover occurs in parallel with the instance of the Alarm activity, i.e., event \hat{e}_{22} occurs between the start (event \hat{e}_{21}) and the complete (event \hat{e}_{23}) of instance of the Alarm activity.

9.2.8 Expanding the High-level Activities and Validating the Model

The abstracted event log L_H hides details on the low-level events. Events in the high-level event log L_H might have been introduced by the approximate abstraction mapping that was obtained through the alignment. As we allow log moves and model moves in the alignment, some parts of the abstracted event log might be based on our assumptions on the process rather than the actual data in the original, low-level event log. We can quantify this error by using the previously introduced measures *fitness* and *matching error*. However, even a small error during the abstraction can be amplified by the discovery algorithm. In other words, we might have misguided the discovery process and resulting process model does reflect our assumption rather than reality. This is clearly undesirable.

To evaluate the quality of the discovered high-level model, we generate a so-called *expanded process model*. We substitute each high-level activity with the DPN representation associated with the activity. In fact, the high-level activities in the discovered process model can be seen as composite activities and the activity patterns as sub-processes. This step depends on the concrete modeling notation.

The expansion of high-level activities with sub-processes cannot be described using solely the language of the process model. When defining a process model through its trace set (i.e., as in Definition 3.2), it is not possible to distinguish whether two high-level activities act_a and act_b are executed in parallel, or their execution is only interleaved. Take, e.g., the two execution traces $\langle act_a, act_b \rangle$ and

$\langle \text{act}_b, \text{act}_a \rangle$, i.e., the two activities a and b can be executed in any order. Assume that activity act_a should be expanded with the sub-process that allows for the low-level behavior $\langle x, y \rangle$ and activity act_b with $\langle z \rangle$. Based on the execution traces, we cannot decide whether the execution sequence $\langle x, z, y \rangle$ (i.e., act_a and act_b are parallel and not interleaved) should be part of the behavior of the expanded process.

However, since the steps are similar in each modeling notation, we focus on the case in which DPNs are employed. For the concrete approach described here, we assume that the discovered model and the activity patterns are provided as DPNs (e.g., as in Figure 9.9).

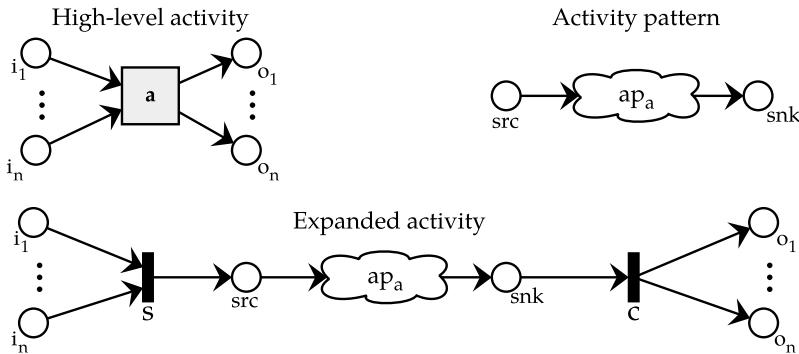


Figure 9.10: Expansion of a single high-level activity a with input places i_1, \dots, i_n and output places o_1, \dots, o_n in the discovered model with the DPN modeling the activity pattern ap_a .

The mapping from high-level activities to activity patterns can be obtained automatically through function hl . Each pattern is mapped to exactly one high-level activity. We replace each transition with the entire corresponding activity pattern. Figure 9.10 illustrates the expansion of a high-level activity a with the process model of the associated activity pattern ap_a . Again, we assume that activity patterns are provided as DPNs with a single source and a single sink place (i.e., workflow nets). With the help of two invisible routing transitions s and c , we connect the source place src and sink place snk of the activity pattern ap_a to the input places i_1, \dots, i_n and output places o_1, \dots, o_n of the replaced high-level activity. The resulting expanded process model describes the behavior of the discovered model in terms of low-level events, i.e., each high-level activity is replaced with a sub process that captures its behavior on a lower abstraction level.

Example 9.10 (Expanded model). Figure 9.11 shows a partially expanded high-level model. We replaced activity Shift in the model shown in Figure 9.9 by the DPN modeling activity pattern ap_a . Routing transitions s and c have been added. Transition s is connected to input place i of activity Shift and to the source

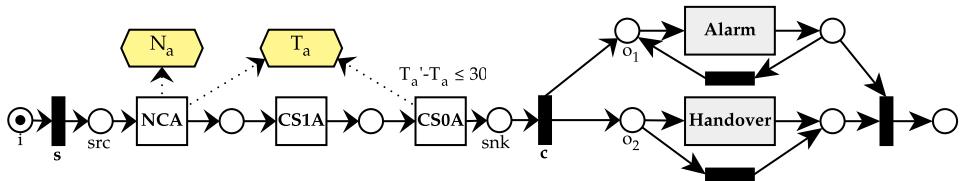


Figure 9.11: A partially expanded high-level model. Activity Shift in the high-level model has been replaced by the DPN that models activity pattern ap_a .

place src of the activity pattern. Transition c is connected to the only sink place snk of the activity pattern and to both output places o_1 and o_2 of the high-level activity Shift. The invisible routing transition s is not strictly necessary. The activity pattern has a single source place. We can simplify the expanded model by fusing places i and src and removing transition s . We use standard reduction rules for this [Mur89].

We can now validate the quality of the expanded model against the low-level input event log. We use existing conformance checking techniques (e.g., alignment-based techniques [AAD12; Man+16c]) that determine the quality (e.g., fitness) of a process model given an event log. We need to validate the model against the original input event log rather than against the abstracted event log. Otherwise, the validation would be biased by the domain knowledge that we introduced in the abstraction. However, when measuring the quality of the expanded model against the original event log, the result is independent from the domain knowledge assumed by using the activity patterns.

9.3 Implementation

The GPD method has been implemented as a set of operators in the RapidMiner extension RapidProM [BLA16].³⁰ Figure 9.12 shows a workflow in RapidProM that implements the entire GPD method.³¹

First, the original event log is imported from a XES file. Then, we import manually created patterns and use discovery methods (e.g., Inductive Miner) to obtain discovered patterns in the sub-process *Identify* (i.e., step 1 of the method). All patterns are composed in the *Compose* sub-process. The *Compose* operator implements step 2 of the method and requires a collection of process models and a composition formula as input. By default all patterns are composed in parallel. We used a sub

³⁰RapidProM is available <http://rapidprom.org>.

³¹The RapidProM workflow can be downloaded under: <https://svn.win.tue.nl/repos/prom/Packages/LogEnhancement/Trunk/data/gpd-workflow.rmp>

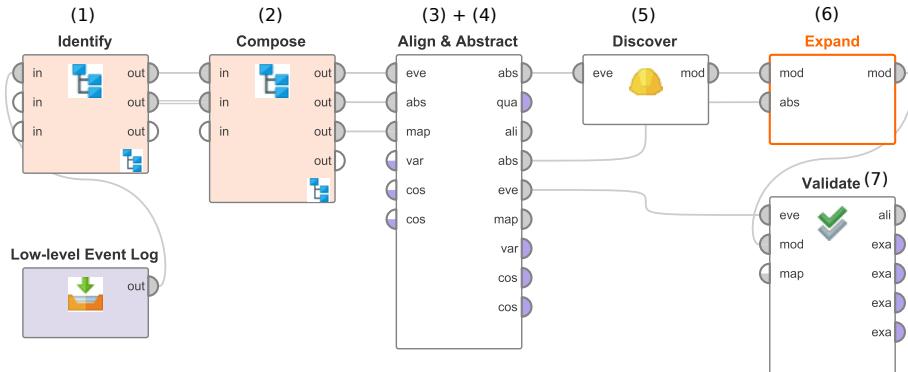


Figure 9.12: A RapidMiner workflow that implements the GPD method. The operators *Identify* and *Compose* are sub-processes, in which the activity patterns are imported or discovered and composed with an operator that implements the composition functions.

process for both steps to ensure that the workflow fits in a figure. We apply the *Align & Abstract* operator, which implements steps 3 and 4 of the method. The required inputs are the event log, the abstraction model and a mapping between the event and transition labels. Afterwards, the standard Inductive Miner operator is used to discover a high-level process model (step 5). Subsequently, this process model is expanded with the new *Expand* operator (step 6). Finally, we use the standard conformance checking capabilities of RapidProM to measure the fitness of the expanded model (step 7).

9.4 Evaluation

We evaluated the effectiveness of the GPD method in a qualitative way by applying it on several real-life data sets. We present the result for three data sets as part of the case studies in Sections 12.5, 13.5 and 14.2. We separated the evaluation of the GPD since the evaluation is based on the insights provided for the case studies (i. e., a qualitative evaluation). The quality of the discovered model highly depends on the kind of domain knowledge that is used during the abstraction. Therefore, the evaluation requires knowledge of the respective background of the event logs that are used. In this section, we evaluate the efficiency of the proposed method. The main complexity of the GPD method stems from the usage of *alignment techniques* to obtain an *optimal abstraction mapping*, i. e., the step presented in Section 9.2.5. We already extensively discussed the efficiency of the alignment computation in Section 5.3. However, the special structure of activity patterns and their composition justifies a separate experiment.

As noted, the computation time of the GPD method is dominated by the align-

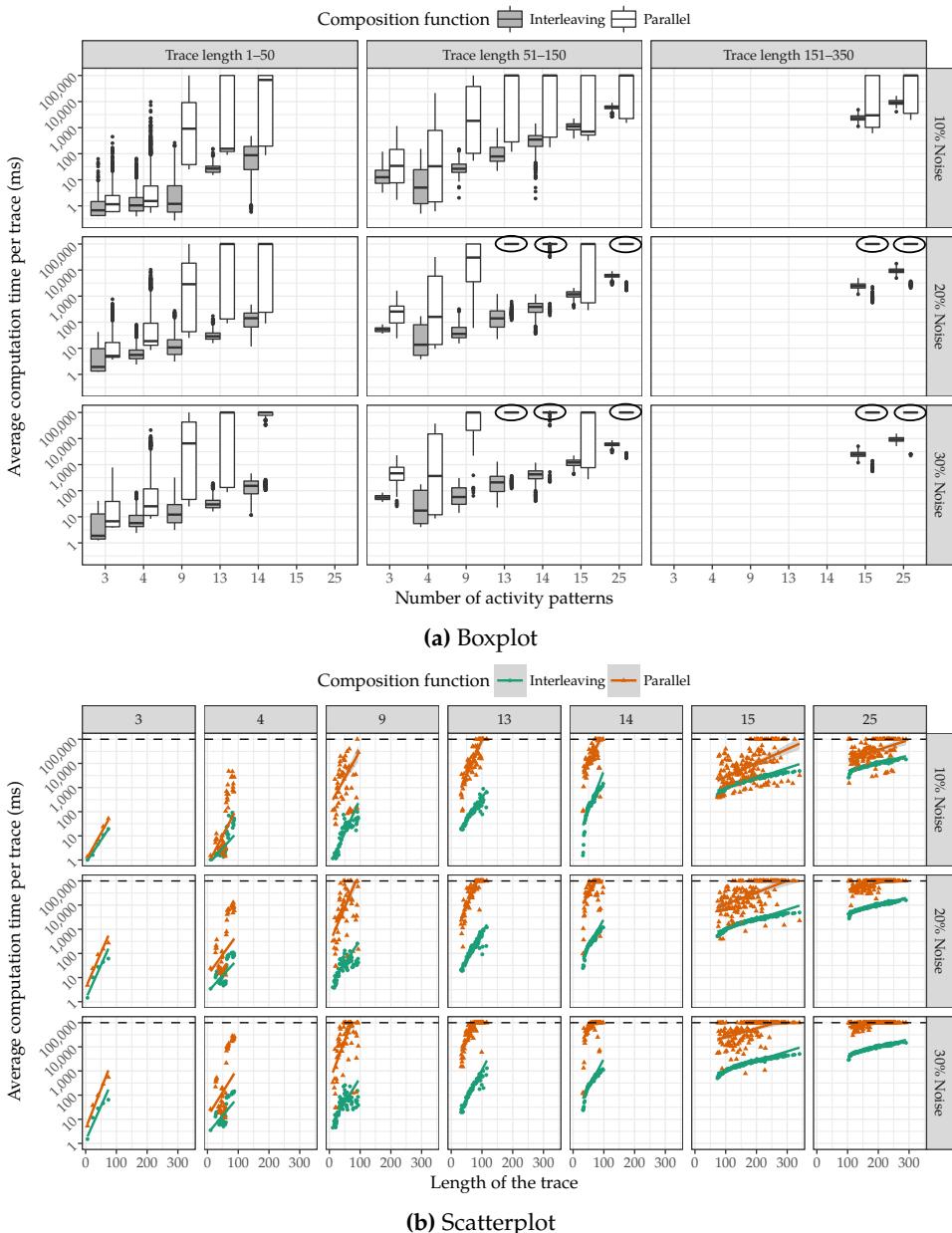


Figure 9.13: Average computation time per trace of the alignment used in the GPD method.

ment computation in step 3 of the method (Section 9.2.5). Computing an optimal alignment has exponential worst-case complexity (cf., Section 5.2.6). The abstraction itself was described in step 4 (Section 9.2.6) and can be computed in time linear to the size of the input trace. Therefore, we evaluated the efficiency of our method by computing alignments for several models and traces. All experiments have been conducted on a standard laptop with 16 GB of memory. We tested the computation time for a set of eight randomly generated process models (25–263 transitions) and event logs (6–340 events per trace) that were previously used in [LM17]. We decomposed each model based on the method described in [MCA14] to obtain between 3 and 25 activity patterns for each model. Then, we compared the performance of using the parallel and the interleaving composition of all activity patterns for increasing levels of noise (10%–30% of swapped events as described in [LM17]) that was injected into the event log.

Figure 9.13 shows the resulting average computation time per trace of our method when applied to each of the event logs. We limited the computation time to 100 seconds (100,000 ms) per trace since we consider this to be a computation time that is still acceptable in practice. The computation time grows exponentially both with increasing length of the traces as well as with the increasing number of activity patterns. Moreover, composing activity patterns in parallel leads to a longer computation time than using the interleaving composition. This is expected due to the large state-space that needs to be explored in case of parallel branches.

Overall, the experiment showed that it is feasible to use the GPD method with up to 25 activity patterns and traces of up to length 350 on occasion of the interleaving composition. When composing all of the patterns in parallel, the computation finished within the maximum of 100 seconds in the cases when less than 13 activity patterns and an event log with less than 20% noise were tested. Conversely, the computation did not finish within 100 seconds for some traces when using the parallel composition. In situations with 13 or more activity patterns, more than 10% noise, or traces that are longer than 150 events our method would have required a longer computation time. Readers should however notice that a composition configuration with all patterns in parallel is the worst case that rarely happens. Usually, an abstraction model is built using a combination of the available composition patterns.

9.5 Related Work

Literature provides several proposals for event abstraction methods. Moreover, there is a large body of work on activity recognition [Liu+16] and event processing [CM12]. We focus on work that is related to the field of process mining (i. e., an explicit process representation is used). We categorize the related work in *unsupervised* and *supervised* event abstraction methods.

Unsupervised methods. There are several approaches for unsupervised event abstraction in the field of process mining. Unsupervised method generally try to determine this relation based on identifying sub-sequences. Among the first proposals for event abstraction in process mining was the Fuzzy Miner by Günther et al. [GA07]. Activities are grouped together based on the significance of the edges between activities. Bose et al. [JA09] proposed to use common patterns based on repeating sequences to abstract events. Later, Günther et al. [GRA10] used agglomerative hierarchical clustering to build a hierarchy of event clusters that can be used for event abstraction. Cook et al. [CKR13] proposed an unsupervised algorithm for activity discovery based on sensor data that is guided by the minimum description length principle. Folino et al. [FGP15] turned event abstraction into a predictive clustering problem and did not assume the notion of an event label in the new approach. Unsupervised abstraction methods do not take existing knowledge into account and should be only applied when no domain knowledge can be used as input. Here, we assume to have some knowledge about those patterns. This knowledge should be leveraged when available because it allows for obtaining more accurate results.

Supervised methods. Approaches for supervised event abstraction assume some knowledge on the relation between low-level events and activities. Methods based on Complex Event Processing (CEP) [CM12] and activity recognition [Liu+16] typically assume a stream of events over which queries are evaluated. When a query is matched a high-level activity is detected. Traditionally, CEP does not consider the notion of process instance (i. e., case) and in case of overlapping queries (e. g., shared functionalities) both high-level activities would be detected. Still, there is some work that uses CEP withing a business process context [Bül+14; HV14; Oli+13; Wei+14]. However, none of these works provides a complete process discovery method based on domain knowledge.

There are also proposals for supervised event abstraction that are more closely related to the field of process mining. Tax et al. [Tax+16b] assume the existence of a labeled training set of traces. Moreover, the approach is limited to processes without concurrent high-level activities. Conditional random fields are used to infer the correct mapping. George et al. [GCW16] also assume a labeled training set of events organized in traces. They apply frequent sequence mining and, then, learn constraints from the events. However, the approach does not deal with noise in the event data. Senderovich et al. [Sen+16b] determine an optimal mapping between sensor data of a real-time locating system and activities based on finding an optimal mapping using integer linear programming. Ferreira et al. [FSR13] assume a complete process model of the high-level activities. They use hierarchical Markov models together with an expectation maximization method to find the mapping between low-level events and the high-level activities in the process model. Later work [FSR14] proposed a different, greedy approach that can better deal with

noise in the event log. Fazzinga et al. [Faz+15] proposed a probabilistic method to find a mapping between an existing high-level model and events. They report in [Faz+15] that their method is feasible only for short traces of less than 30 events.

Most related to our work are the methods developed by Baier et al. [Bai+15; Bai15; BMW14]. Again, the methods assume knowledge about a single high-level model for the overall process. The goal is to automatically discover the relation between events and activities. Therefore, these methods are mainly targeting the situation where the process is assumed to be well known. The proposed methods use clustering methods and heuristics when challenged with event logs from processes that feature *concurrent* high-level activities and *noise* (i. e., erroneous or missing events). A later proposal using constraint programming approach (cf., [Bai15]) only considers the control-flow perspective, i. e., rules based on data are not supported. Begicheva et al. proposed a method for supervised log abstraction in [Beg17]. Their proposed method requires a mapping from low-level activities to high-level activities as input. The method abstracts low-level events by directly replacing the low-level events with the corresponding high-level activities. However, it works only for *non-cyclic* high-level process models, i. e., high-level activities may not be repeated, and it does not consider *shared functionality* and *noise* in the low-level event log.

To conclude, none of the related work tackles the problem of systematically guiding process discovery methods towards discovering a better model by using supervised event abstraction that can deal with *noise*, *concurrency*, *shared behavior*, and *multiple process perspectives*.

9.6 Conclusion

We presented a new method that uses multi-perspective *event abstraction* based on domain knowledge to guide process discovery towards better results.

9.6.1 Contribution

The GPD method uses *multi-perspective activity patterns* that encode assumptions on how high-level activities manifest themselves in terms of recorded low-level events, i. e., we use activity patterns to encode domain knowledge on the relation between low-level events and high-level activity instances. We use this domain knowledge to establish the functional perspective of the process. An abstracted event log is created on the basis of an alignment between activity patterns and the low-level event log. We use the abstracted event log to discover a high-level process model. Since the high-level process model is not defined over the same activities as recorded in the low-level event log, we added two further steps to the method. First, we expand the high-level activities of the high-level process model with the activity patterns. Then, we use the expanded process model, which is

defined over the same activities as in the low-level event log, to validate the quality of the discovered process model. We evaluated the efficiency of the method and showed that it can be applied to examples of moderate size and complexity that are often encountered in practice. Later in Sections 12.5, 13.5 and 14.2, we demonstrate that the GPD method could be used for real-life event logs using three case studies.

9.6.2 Limitations

We acknowledge that there are some limitations to our method.

- The performance of the method highly depends on the quality of the activity patterns used. We introduced the expansion step in the GPD method to limit the risks of using low-quality activity patterns, i.e., patterns that do not properly capture the real execution of the process. The expanded process model can be validated on the original event log. Hence, quality problems can be detected.
- If a sequence of events fits two activity patterns perfectly, one of them will be chosen arbitrarily. This is due to the particularities of the underlying alignment method employed. The cost-based alignment techniques that we use to determine the optimal mapping chooses an arbitrary pattern in case there are multiple mappings with the same cost.
- Since the GPD method relies on alignment techniques, it requires a lot of resources. Computing alignments is a computationally expensive operation and may be infeasible for event logs with very long traces and many activity patterns. However, approximate and more efficient alignment techniques are being developed, e.g. [Don+17; TC16].

9.6.3 Future Work

There are several directions for future work that are worthwhile exploring.

- Obtaining suitable activity patterns for abstraction may be difficult in situations with little domain knowledge. Methods to derive activity patterns automatically from an event log could help to provide an initial set of likely patterns. Such methods could be, e.g., based on pattern mining techniques such as local process model discovery [Tax+16a] or episode discovery [LA15]. Initial results using local process model discovery have already been presented in [MT17].
- Work on decomposing or approximating the alignment computation for abstraction models could help to alleviate the performance problems of the method. Since an abstraction model imposes a specific structure on the

composition of activity patterns, we believe that tailor-made decomposition method could be created.

- As mentioned, a limitation to our method is that if there are multiple optimal alignments for a sequence of events, i.e., multiple different instantiations of activity patterns could explain the observed behavior, then, one of them will be chosen arbitrarily. A prioritization of activity patterns used during the alignment computation could be introduced. Moreover, it would be possible to introduce a simple heuristic that minimizes the number of pattern instantiations by introducing a small cost for instantiating a pattern to the alignment technique.

10 Enhancing Models with Overlapping Decision Rules

This chapter describes a method to enhance an existing process model with decision logic that further constrains the routing of process instances: we present a novel **decision mining** [RA06] method, which discovers potentially **overlapping decision rules** based on an event log. Overlapping (i.e., non mutually-exclusive) decision rules are often encountered in practice since contextual information relevant for the actual decision making is unavailable. We introduce the general idea of decision mining in Section 10.1 and motivate the need to overlapping decision rules in Section 10.2. Then, we present our decision mining method in Section 10.3 and evaluate it using real-life event logs in Section 10.4. We review related work in the fields of process mining, classification, and rule learning in Section 10.5. Finally, we conclude this chapter in Section 10.6.

10.1 Introduction to Decision Mining

To introduce decision mining we first briefly motivate the need for decision rules in Section 10.1.1 and, then, presenting the state-of-the-art of decision mining techniques in Section 10.1.2.

10.1.1 Decision Rules

Generally, process models describe the activities (i.e., units of work) and their dependencies in a graphical representation, which specifies the order of activities in the process execution. However, during the execution of any non-trivial processes, next to the ordering of activities, *decisions* between multiple alternative activities need to be made. Those choices can be modeled in process models as so-called *decision points*. Take, e.g., the process model shown in Figure 10.1. When place p_2 is marked with a token, then, an exclusive choice between the execution of transitions τ_1 and τ_2 needs to be made. The patient is either admitted to the emergency ward (τ_1) or discharged without admission (τ_2). We denote place p_2 as decision point, which specifies the alternatives available.³² Process decision can be based on several

³²Here, we consider only the simple form of decision points. Generally, more complex decision points can be defined with Petri nets, e.g., when multiple tokens need to be synchronized, then, multiple places can be involved in the same decision.

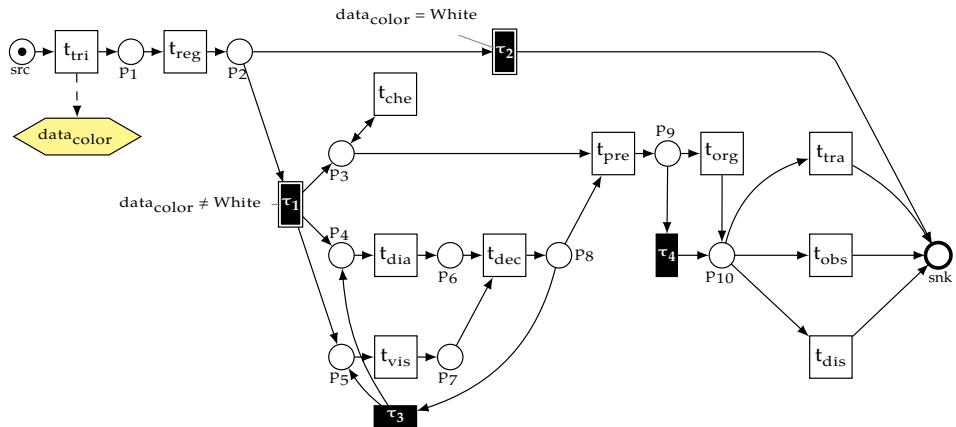


Figure 10.1: A decision rule that governs the routing of process instances at the decision point p_2 of the hospital process.

Decision point p_2		Enter Advanced Mode	Show DRD	
U	Input +	Output +	Annotation	
	color string	transition string		
1	"White"	" τ_2 "	-	
2	not("White")	" τ_1 "	-	

Figure 10.2: Decision table for the rules at place p_2 as specified by the DMN standard.

perspectives on the process: available resources might influence the routing of cases (resource perspective), deadlines might trigger the routing of exception handling activities (time perspective), and as shown in Figure 10.1 data values gathered during the process execution might affect the routing (data perspective).

An important challenge when using process models is to understand the decision that need to be made in a process, and the conditions under which certain alternative activities are performed. Awareness is increasing that modeling and analyzing decisions made during the process execution is of major interest in BPM. See, e.g., the interest in the Decision Model and Notation (DMN) standard [DMN16] supported by vendors such as Signavio and Camunda. Camunda provides a free online editor dmn.js for DMN decision tables.³³ Figure 10.2 shows the decision rules regarding transitions τ_1 and τ_2 as a decision table created with dmn.js.

³³dmn.js can be downloaded from <https://github.com/bpmn-io/dmn-js>

10.1.2 Mining Decision Rules

There are several *decision mining* methods to learn the decision logic of a process by using event logs. Most existing work [BBW16; Gri+04; LA13b; RA06; Roz+09] employs classification techniques to determine decision rules. A crucial assumption for decision mining is that the events used for decision mining contain process data, which was available when the activity was performed (i. e., attributes), and that this process data influenced the routing decision.

Table 10.1: Excerpts of 4 traces of an event log L_{dec} recorded by the hospital process.

(a) Trace σ_{d1}			(b) Trace σ_{d2}		
id	activity	color	id	activity	color
e_{d10}	Triage	White	e_{d20}	Triage	Red
e_{d11}	Register		e_{d21}	Register	
			e_{d22}	Check	
			
(c) Trace σ_{d3}			(d) Trace σ_{d4}		
id	activity	color	id	activity	color
e_{d30}	Triage	Orange	e_{d40}	Triage	Green
e_{d31}	Register		e_{d41}	Register	
e_{d32}	Check		e_{d42}	Check	
...	

Decision mining techniques transform the rule discovery problem into a classification problem. The considered features are the data attributes that have been observed before the decision point was reached and the classes to be predicted are the alternative activities that may occur. The challenge decision mining methods face is to build a suitable set of training instances based on an event log.

Take, for instance, the four traces recorded in Table 10.1. The rule discovery problem for decision point p_2 and transitions τ_1 and τ_2 is transformed into a classification problem that tries to predict the two classes: τ_1 and τ_2 based on previously recorded attributes values (e. g., color in Table 10.1). In case the activities involved in a decision point are visible activities (e. g., decision point p_{10} with activities Transfer, Observe, and Check) and the event log is fully fitting the process model, then, training instances are easy to obtain. For each occurrence of an event referring the the activity name Transfer a training instance associated with the class Transfer is built. The attribute values of that training instance are determined by taking the latest values of all attributes.

However, often event logs contains *noise*, i. e., deviations from the behavior pre-

scribed by the process model. Decision mining methods require process traces (i. e., execution sequence of the process model). However, log traces may be not fitting and may not contain the executions of routing activities (e. g., invisible transitions in a DPN). In these cases it is not straightforward to determine whether a training instance should be created for an event if so for which transition. For example, for decision point p_2 in Figure 10.2 it is not obvious how to build the training instances for the invisible activities τ_1 and τ_2 because, by definition, no event records their execution.

Therefore, decision mining techniques need to relate recorded events reliably to activities of the process model. This challenge has been addressed by using heuristics [RA06] and by using alignment techniques [LA13b] such as the one presented in Chapter 5.

In case of event log L_{dec} , an alignment of the traces to the Petri net of the hospital process (i. e. the model shown in Figure 3.2 without guards and variables) would indicate that transition τ_2 needs to be executed in trace σ_{d1} to end the process, i. e., an optimal alignment is:

$$\gamma_{\sigma_{d1}} = \langle (e_{d10}, (t_{tri}, \emptyset)), (e_{d11}, (t_{reg}, \emptyset)), (\gg, (\tau_2, \emptyset)) \rangle.$$

The optimal alignments obtained for the other three traces would indicate that transition τ_1 needs to be executed:

$$\begin{aligned}\gamma_{\sigma_{d2}} &= \langle (e_{d20}, (t_{tri}, \emptyset)), (e_{d21}, (t_{reg}, \emptyset)), (\gg, (\tau_1, \emptyset)), (e_{d22}, (t_{che}, \emptyset)), \dots \rangle; \\ \gamma_{\sigma_{d3}} &= \langle (e_{d30}, (t_{tri}, \emptyset)), (e_{d31}, (t_{reg}, \emptyset)), (\gg, (\tau_1, \emptyset)), (e_{d32}, (t_{che}, \emptyset)), \dots \rangle; \\ \gamma_{\sigma_{d4}} &= \langle (e_{d40}, (t_{tri}, \emptyset)), (e_{d41}, (t_{reg}, \emptyset)), (\gg, (\tau_1, \emptyset)), (e_{d42}, (t_{che}, \emptyset)), \dots \rangle.\end{aligned}$$

Based on those alignments, we can obtain the following multi-set of training instances. Each instances is composed of the latest observed attribute values in the event log and the outgoing transition that was observed on the decision point p_2 :

$$\begin{aligned}[(color \mapsto White), \tau_2], \\ [(color \mapsto Red), \tau_1], [(color \mapsto Orange), \tau_1], [(color \mapsto Green), \tau_1]\end{aligned}$$

Applying a standard classification technique such as the decision tree learner C4.5 [Qui93] (cf., Definition 2.13) on this set of training instance yields the decision tree shown in Figure 10.3.

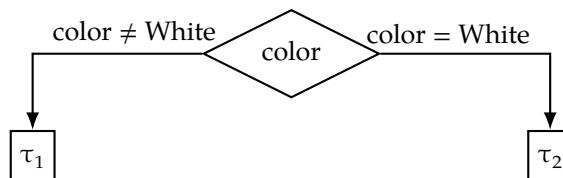


Figure 10.3: Decision tree learned from a multi-set of training instances.

Decision mining methods [LA13b; RA06] transform such a decision tree into decision rules that restrict the possibility to execute transitions τ_1 and τ_2 . The problem of decision mining can be regarded as that of discovering a DPN that characterizes the process: given a Petri net, we aim to discover the variables, write operations and guards of a DPN. In this case a variable $\text{data}_{\text{color}}$ written by transition t_{tri} and the following guard expressions are introduced:

$$\begin{aligned} gd(\tau_1) &= \text{data}_{\text{color}} \neq \text{White} \\ gd(\tau_2) &= \text{data}_{\text{color}} = \text{White}. \end{aligned}$$

This yields the DPN that is shown in Figure 10.1. In case the decision tree contains multiple conditions (i. e., its depth is greater than 1) or multiple leafs for the same transition, the decision rule for a transition t is built as follows. For each leaf, a rule is built by taking the conjunction of all conditions represented by those nodes that are encountered on a path from the leaf node up to the root node [LA13b]. Then, the rules for all leafs that are labeled with transition t are combined in disjunction. This way, mutually-exclusive rules for each transition are obtained.

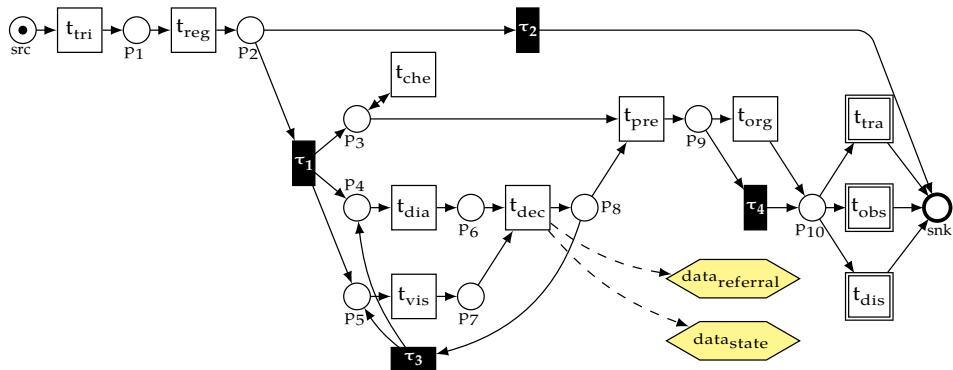
10.2 Motivation for Overlapping Decision Rules

Traditionally, decision mining methods [Gri+04; LA13b; RA06; Roz+09] use decision tree learning techniques such as C4.5 [Qui93] to determine the rules governing the process execution based on event logs. As explained in Section 10.1.2, the recorded attribute values are used as *feature*, while the choice between the transitions is used as *target class*. Then, mutually exclusive rules for each activity are built using the obtained decision tree. By doing so, the choice at the decision point is entirely determined by the values of the attributes.

Existing decision mining techniques for exclusive choices rely on the strong assumption that the rules attached to the alternative activities of a exclusive choice need to be *mutually exclusive*. However, in when dealing with real-life event logs, this is not always the case. Root causes for non mutually-exclusive rules may be (A) business rules that are non-deterministic and (B) missing information on the attributes that determine the decision in the event log.

Regarding (A), business rules may be non-deterministic due to conflicting rules or due to missing contextual information. Often, this ambiguity “*cannot be solved until the business rule is instantiated in a particular situation*” [RW02]. For example, decisions taken by process workers may depend on contextual factors, which are not encoded in the system and, thus, cannot be captured in automatic decision support.

Regarding (B), even if all those factors are encoded in the system event logs extracted from systems are often incomplete [BMA13]. Without complete information in the event log, the mutually exclusive rules underlying the decision-making



(a) Process model of the hospital process.

Transition	Guard expression
t_{tra}	$data_{referral} \neq Home$
t_{obs}	$data_{referral} \neq Home \wedge data_{state} = Unstable$
t_{dis}	$data_{referral} = Home \vee (data_{referral} \neq Home \wedge data_{state} = Stable)$

(b) For some assignments two out of the three guards expressions are fulfilled.

Figure 10.4: Overlapping decision rules at decision point p_{10} of the hospital process model.

cannot be discovered. Hence, the assumption of mutually exclusive rules is often not met in the reality of a process mining analysis with incomplete data.

For example, consider the process model shown in Figure 10.4, which depicts guard expressions for output transitions of the decision point p_{10} of the hospital process³⁴. At decision point p_{10} only one out of the three alternative transitions t_{tra} , t_{obs} , and t_{dis} can be taken. Thus, the decision point models an *exclusive choice*³⁵, still the specified guard expressions are overlapping. For process instances with the variable assignment ($data_{referral} \mapsto$ Tertiary, $data_{state} \mapsto$ Unstable) patients may be transferred (t_{tra}) or allocated to the observation ward (t_{obs}).

This means that for a decision point more than one of multiple activities may be executed under the same condition, i.e., when the same attribute values have been observed beforehand. The actual decision between both activities is not specified. It might depend on an unavailable contextual factor. Some patients may be transferred even when their state is unstable because the appropriate means of transport (e.g., an helicopter) is available. The availability of the means of transport is not recorded in an information system or the information is not available for the analysis.

³⁴In Figure 10.1, we refined the basic guard expressions introduced in Figure 3.3 with an additional variable $data_{state}$ for the sake of illustrating our decision mining technique.

³⁵Note that p_{10} does **not** model an inclusive choice that would allow the execution of multiple alternatives. Inclusive choices may also be associated with overlapping rules, but this is considered out of scope here.

State of the art techniques for decision mining [Gri+04; LA13b; RA06] cannot discover this class of rules. For instance, current techniques fail to discover the decision rule in Figure 10.1.

10.3 Discovery of Overlapping Decision Rules

We describe the technique that discovers *overlapping rules* in those cases that the underlying observations are characterized better by such rules. The method is published in [Man+16b]. We aim to deliberately trade the *precision* of mutually-exclusive rules, i. e., only one alternative is possible, against *fitness*, i. e., overlapping rules that are less often violated. Before presenting the decision mining technique, we clarify the required input and describe the parameters of our method.

10.3.1 Parameters and Assumptions on the Input

As depicted in Figure 10.5 the input to our method is an event log containing information about process executions and a process model, e. g., a Petri net or process model that can be converted to a Petri net (e. g., BPMN, EPC, C-Net). Moreover, we support three parameters: the set of *selected attributes*, *min instances*, and the *merge ratio*:

- Parameter *selected attributes*, $V_L^{\text{SEL}} \subseteq V_L$, determines which attributes of the event log are used as features of the classification technique.
- Parameter *min instances*, $mi \in \mathbb{N}$, controls the level of detail of the rules that are to be discovered. The lower this parameter, the more complex rules are returned since the decision tree learner may return larger trees.
- Parameter *merge ratio*, $mr \in [0, 1]$, controls the error that may be introduced by using majority votes of the underlying decision tree to create overlapping rules.

For the sake of a simpler presentation, we make six assumptions regarding the input event log $L = (E, \Sigma, \#, \mathcal{E})$.

1. All recorded process instances are *perfectly fitting* with regard to the process represented by the Petri net (P, T, F).
2. The event log contains events for invisible routing transitions.
3. The executed transition in the Petri net can be uniquely determined for each event $e \in E$ of the event log, i. e., we are given a **transition mapping function** $trans : E \rightarrow T$ with which each event can be uniquely mapped onto a single transition of the model.
4. All the attributes $a \in V_L$ of the event log are selected for decision mining and, thus, we assume $V_L = V_L^{\text{SEL}}$.
5. Events write attributes consistently, i. e., if an event e writes an attribute $a \in \#(e)$, then all events $e' \in E$ corresponding to the same transition, i. e.,

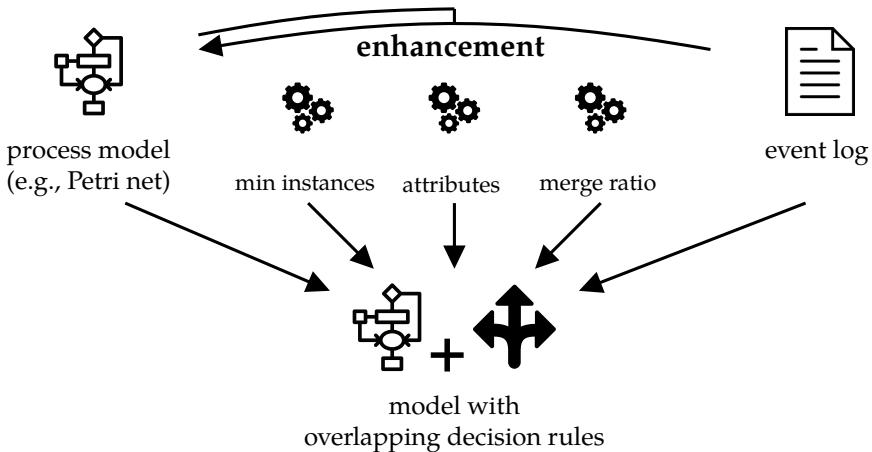


Figure 10.5: Overview of the proposed overlapping decision mining method.

$\text{trans}(e) = \text{trans}(e')$, also write attribute $a \in \#(e')$.

6. We assume to be given the **initial values** $\text{INIT} : \mathcal{U}_{\text{dom}}^{\text{L}}$ of each selected attribute $a \in V_L$.

Those assumptions do not limit the applicability of the method. For any event log that does not meet these requirements, we can transform the event log to the closest event log matching those requirement. For example, we can use alignment-based techniques as presented in Chapter 5 to meet the first three assumptions. We create an event log that is the closest to a fully compliant event log and includes artificial events for invisible routing transitions. Then, we remove all unselected attributes from the event log. Regarding the last two assumption, we show later in Section 10.3.4 how to handle event logs that do not meet those requirements.

10.3.2 Overall Decision Procedure

We describe the overall decision procedure, which abstracts from the method with which concrete decision rules are discovered. The overall procedure is adopted from the approach proposed in [LA13b]. The goal is to build a DPN $N = (P, T, F, V_P, \text{dom}, \text{in}, wr, gd)$ based on an event log $L = (E, \Sigma, \#, \mathcal{E})$ and a Petri net (P, T, F) .

Discovering Write Operations and Variables of the DPN

We build the set of variables $V_P \subseteq V$ with their domain dom and discover the write operations wr based on the event log as follows:

1. We build the set V_P of variables of the DPN based on the set V_L of selected attributes. For each selected attribute a corresponding variable $v \in V_P$ is

created. For a less verbose presentation, we assume in the remainder of this section that $V_P = V_L$, i. e., the DPN is defined directly over the set of selected attributes. The domain dom of each attribute can be determined, e. g., by taking all values that occur in the event log³⁶ or information available in meta-data of the XES format [IEECCIS16].

2. For each transition $t \in T$ of the Petri net if there exists an event $e \in E$ with $trans(e) = t$, then variables $v \in V_P$ are written with the values of the corresponding attributes:

$$\forall_{v \in V_P} \forall_{t \in T} (\exists_{e \in E} (trans(e) = t \wedge v \in dom(\#(e))) \implies v \in wr(t))$$

After having discovered the write operations wr , domain dom and variables V_P , we describe how to discover the guard expressions gd in the next section.

Discovering Guard Expressions

For each decision point $p \in P$ (i. e., places with two or more output transitions) we construct a set of *observation instances* that are related to place p . Those observation instances are later used to train the decision tree classifier that is used to discover the guards of the DPN. We use \biguplus to generalize the sum of multi-sets to the sum of a set of multi-sets, i. e., $\biguplus_{x \in \{X, Y, Z\}} x = Y \uplus Y \uplus Z$.

Definition 10.1 (Observation Instances). Let U be a universe of possible values. Let $V_L \subseteq V$ be a set of attributes. Let $dom(v) \in \mathbb{P}(U)$ be the domain of a variable $v \in V_L$. Let $L = (E, \Sigma, \#, \mathcal{E})$ be a perfectly fitting event log. Let (P, T, F) be a Petri net. Function $obsInst_L : P \rightarrow \mathbb{B}(\mathcal{U}_{dom}^L \times T)$ returns the multi-set of observation instances³⁷ for a place $p \in P$:

$$obsInst_L(p) = \biguplus_{\{e \in E \mid trans(e) \in p\bullet\}} [(INIT \oplus latest(e), trans(e))] \quad \diamond$$

For each event $e \in E$ that refers to an output transition of p , i. e., $trans(e) \in p\bullet$, the set of observation instances of p contains an instance $(w, t) \in obsInst_L(p)$, where $w \in \mathcal{U}_{dom}^L$ are the *observed values of the attributes*, and t the *observed transition execution*. The values of w are obtained by taking overriding the latest observed value for the attributes in preceding events and the initial values.

Example 10.1 (Building the set of observation instances). Take the event log shown in Table 10.2 and the Petri net of the hospital process. Since we assumed that the event log is fully fitting and contains events for invisible routing transitions,

³⁶This can be computed in a single pass through the event log.

³⁷We already introduced the multi-set of observation instances in Definition 2.12.

Table 10.2: Excerpts of 4 traces of the event log L_{dec} that have been extended to meet our assumptions: fully fitting and including events for invisible transitions.

(a) Trace $\bar{\sigma}_{d1}$			(b) Trace $\bar{\sigma}_{d2}$		
id	activity	color	id	activity	color
e_{d10}	Triage	White	e_{d20}	Triage	Red
e_{d11}	Register		e_{d21}	Register	
e_{d12}	τ_2		e_{d22}	τ_1	
			e_{d23}	Check	
			
(c) Trace $\bar{\sigma}_{d3}$			(d) Trace $\bar{\sigma}_{d4}$		
id	activity	color	id	activity	color
e_{d30}	Triage	Orange	e_{d40}	Triage	Green
e_{d31}	Register		e_{d41}	Register	
e_{d32}	τ_1		e_{d42}	τ_1	
e_{d33}	Check		e_{d43}	Check	
...	

the set of observation instances can be built in one pass through the event log. For example, for trace $\bar{\sigma}_{d1}$, one instance $((color \rightarrow White), \tau_2)$ is added since $latest(e_{d12}) = (color \rightarrow White)$ and $trans(e) = \tau_2$. Please note that multiple instances can be added for one trace when the process contains loops. The set of observation instances for place p_2 is:

$$\begin{aligned} obsInst_{L_{dec}}(p_2) = [& ((color \rightarrow White), \tau_2), \\ & ((color \rightarrow Red), \tau_1), \\ & ((color \rightarrow Orange), \tau_1), \\ & ((color \rightarrow Green), \tau_1)] \end{aligned}$$

This corresponds to the set that was informally introduced in Section 10.1.2.

Algorithm 6 describes the overall discovery method for the entire Petri net. Using the observation instances $obsInst_L(p)$, we build the guard estimation function $est_p : T \rightarrow EXPR_{V_p}$ for each decision point $p \in P$ through function $buildEstimator$. Function returns a set of potentially overlapping decision rule for each output transition of place p . We introduce the implementation of function $buildEstimator$, which forms the main ingredient of our method, in Section 10.3.3. Having obtained the guard function, we assign each transition the conjunction of all rules obtained

Input: Petri net $((P, T, F))$, perfectly fitting event log $(L = (E, \Sigma, \#, \mathcal{E}))$, minimum instances (mi) , merge ratio (mr)

Result: Guard function of the DPN (gd)

```

1 foreach  $p \in P$  s.t.  $|p\bullet| > 1$  do
2    $OI_p \leftarrow obsInst_L(p)$ 
3    $est_p \leftarrow buildEstimator(p, OI_p, mi \cdot |OI_p|, mr)$ 
4 foreach  $t \in T$  do
5    $gd(t) \leftarrow \text{true}$ 
6   foreach  $p \in \bullet t$  do
7      $gd(t) \leftarrow gd(t) \wedge est_p(t)$ 
8 return  $gd$ 
```

Algorithm 6: Procedure that discovers the guards of a DPN based on an event log.

from its input places. All guard expressions need to be fulfilled for a transition to be enabled.

10.3.3 Building Overlapping Guard Expressions

The construction of the observation instances with function $obsInst_L$ and the overall discovery procedure share similarities with the previous work [LA13b]. However, our actual discovery technique considerably differs in how the rules are obtained through function $buildEstimator$. Our contribution is a new algorithm that discovers guards that may be partially overlapping, i.e., two or more transitions may be enabled for some state reachable in the DPN. The technique builds an initial decision tree based on observations from the event log. Then, for each decision tree leaf, the wrongly classified instances are used to learn a new decision tree possibly leading to new rules. These new rules are used in disjunction with the initial rules, which yields overlapping rules of the form $expr_1 \vee expr_2$.

Next, we describe how to define function $buildEstimator$, which discovers overlapping guards for place p given the observation instances $OI = obsInst_L(p)$, the set of variables V_p , and two user-defined parameters: the minimum number of instances mi and the merge ratio mr . Algorithm 7 describe the procedure that builds a guard estimation function $est_p : T \rightarrow \text{EXPR}_{V_p}$ for place p .

Initially all transitions are assigned a placeholder value $void$. For notational convenience, we assume that $void$ does not influence the result of a boolean expression, i.e., $\forall_{expr \in \text{EXPR}_V} ((expr \vee void) = (expr \wedge void) = expr)$ (line 1).

First, a **base decision tree** $baseTree = buildTree_{mi}(OI)$ is built based on all observation instances.³⁸ Each leaf $(expr, t)$ of the base decision tree corresponds to a rule $expr$ that predicts a single transition t as outcome (line 2). Each discovered rule is

³⁸Remember that $buildTree_{mi}$ returns the leafs of a decision tree trained with C4.5, cf. Definition 2.13.

```

Input: Place (p), variables ( $V_p$ ), observation instances (OI), minimum instances ( $mi$ ),
merge ratio ( $mr$ )
Result: Guard Estimation Function ( $est_p$ )
1  $\psi : T \rightarrow \text{EXPR}_{V_p} \cup \{\text{void}\}$  s. t.  $\forall t \in p \bullet : \psi(t) \leftarrow \text{void}$ 
2 foreach leaf:  $(expr, t) \in buildTree_{mi}(OI)$  do
3    $\psi(t) \leftarrow \psi(t) \vee expr$ 
4    $WI \leftarrow \{(w, t') \in OI \mid t \neq t' \wedge expr \text{ evaluates to true for } w\}$ 
5    $\overline{mi} \leftarrow mi \cdot \frac{|WI|}{|OI|}$ 
6    $subTree \leftarrow buildTree_{\overline{mi}}(WI)$  // tree based on wrong instances
7   if  $|subTree| > 1$  then
8     foreach subLeaf:  $(subExpr, t') \in subTree$  do
9        $\psi(t') \leftarrow \psi(t') \vee (expr \wedge subExpr)$ 
10  else
11     $\{(true, t')\} \leftarrow subTree$  // majority vote for  $t'$ 
12    if  $|WI| > mi$  and  $(\psi(t') \neq \text{void} \text{ or } \frac{|\{(w, t) \in WI \mid t \neq t'\}|}{|WI|} < \epsilon)$  then
13       $\psi(t') \leftarrow \psi(t') \vee (expr)$ 
14 foreach  $t \in p \bullet$  do
15   if  $\psi(t) = \text{void}$  then
16      $est_p(t) \leftarrow \text{true}$  // no rule could be found
17   else
18      $est_p(t) \leftarrow \psi(t)$ 
19 return  $est_p$ 

```

Algorithm 7: Procedure *buildEstimator*, which a guard estimator that computes partly overlapping guards.

added to the helper function $\psi(t)$ in disjunction to the already discovered rules (line 3). In case no rule has been discovered yet, the placeholder value *void* has no influence on the result. For each leaf of the base decision tree $leaf = (expr, t)$, we extract those instances WI that have been wrongly classified by the base classifier: WI contains all those instances $(w, t') \in OI$ for which the predicted transition t' is different from the transition t in leaf (line 4).

For each leaf, we build an **new decision tree** $subTree$ based on those WI that have been wrongly classifier by the base classifier. Since the size of WI can be significantly smaller than that of OI , we scale down the parameter mi to $\overline{mi} = mi \cdot \frac{|WI|}{|OI|}$ (lines 5-6). The idea is that the second decision tree $subTree$ can *further discriminate* between the observed transitions among the wrongly classified instances. The second decision tree possibly introduces partial overlap with the existing rule. Please note that a new decision tree can be discovered since we reduce parameter mi . Moreover, the base decision tree might have been pruned to prevent overfitting. There are two

possible cases:

1. a decision tree with more than one leaf is found: $|subTree| > 1$;
2. only a single-leaf decision tree ($true, t'$) representing a majority vote is found.

In the **first case**, we build rules for each leaf $subLeaf = (subExpr, t') \in subTree$ by taking the conjunction of the rule $expr$ from leaf $subLeaf$ of tree $baseTree$ and the newly discovered rule $subExpr$. We obtain the new guard of transitions t' by adding newly discovered rules in disjunction to the existing ones (lines 8-9).

In the **second case** the decision tree represents a majority vote, i. e., the transition that was most often observed within the wrong instances is predicted. In this case we add rule $expr$ to the existing guard $\psi(t')$, but only if two conditions are met to avoid overfitting the data and to avoid introducing poorly fitting rules (line 12):

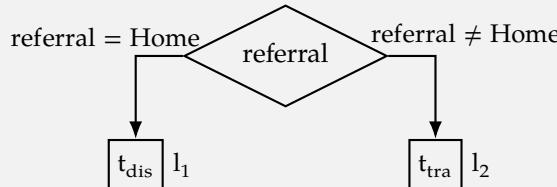
1. the set of wrong instances is larger than the original user-specified number of minimum instances mi and
2. the base decision tree already contained a rule regarding transition t' or the fraction of observation instances in WI referring to t' is larger than the user-specified merge ratio mr .

The second condition is added to prevent introducing a rule for leafs that represent a bad majority vote (e. g., leafs in which only 51% of the instances refer to transition t'). Finally, we assume that all transitions for which no rule could be found , i. e., that are still mapped to *void*, are always enabled (lines 14-18).

Example 10.2 (Estimating Partly Overlapping Guard Expressions). Assume we want to use Algorithm 7 to estimate the guards of the process model in Figure 10.1. We obtained the following multi-set of observation instances for place p_{10} :

$obsInst_{L_{dec}}(p_{10}) = [((referral \mapsto Home, state \mapsto Stable), t_{dis})^{100}, ((referral \mapsto Ward, state \mapsto Stable), t_{dis})^{10}, ((referral \mapsto Ward, state \mapsto Stable), t_{tra})^{50}, ((referral \mapsto Ward, state \mapsto Unstable), t_{tra})^{50}, ((referral \mapsto Ward, state \mapsto Unstable), t_{obs})^{10}, ((referral \mapsto Tertiary, state \mapsto Stable), t_{dis})^5, ((referral \mapsto Tertiary, state \mapsto Stable), t_{tra})^{10}, ((referral \mapsto Tertiary, state \mapsto Unstable), t_{obs})^5, ((referral \mapsto Tertiary, state \mapsto Unstable), t_{tra})^{10}]$

The base decision tree returned by C4.5 is:



Based on this base decision tree, we obtain the initial expressions:

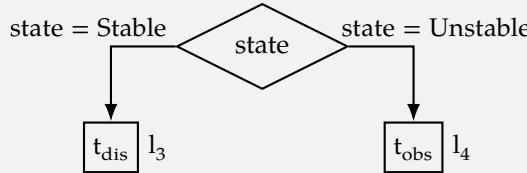
$$\psi(t_{\text{dis}}) \leftarrow \text{data}_{\text{referral}} = \text{Home}$$

$$\psi(t_{\text{tra}}) \leftarrow \text{data}_{\text{referral}} \neq \text{Home}$$

Then, we collect the set of wrongly predicted instances for leaf l_1 , i. e., all instance that fulfill the associated expression ($\text{referral} = \text{Home}$) but were not recorded for transition t_{dis} . This set $WI_{l_1} = []$ is empty, i. e., there are not such instances. Thus, no subtree can be discovered for leaf l_1 . However, for the other leaf l_2 there are multiple wrongly classified instances:

$$\begin{aligned} WI_{l_2} = & [((\text{referral} \mapsto \text{Ward}, \text{state} \mapsto \text{Stable}), t_{\text{dis}})^{10}, \\ & ((\text{referral} \mapsto \text{Ward}, \text{state} \mapsto \text{Unstable}), t_{\text{obs}})^{10}, \\ & ((\text{referral} \mapsto \text{Tertiary}, \text{state} \mapsto \text{Stable}), t_{\text{dis}})^5, \\ & ((\text{referral} \mapsto \text{Tertiary}, \text{state} \mapsto \text{Unstable}), t_{\text{obs}})^5]. \end{aligned}$$

We use set WI_{l_2} to build the following subtree:



Now, we combine the expression obtained for leaf l_2 of the base decision tree with the newly discovered expressions in conjunction and combine the result in disjunction to any existing expressions:

$$\begin{aligned} \psi(t_{\text{dis}}) &\leftarrow \psi(t_{\text{dis}}) \vee (\text{data}_{\text{referral}} \neq \text{Home} \wedge \text{data}_{\text{state}} = \text{Stable}) \\ &= (\text{data}_{\text{referral}} = \text{Home} \vee (\text{data}_{\text{referral}} \neq \text{Home} \wedge \text{data}_{\text{state}} = \text{Stable})) \\ \psi(t_{\text{obs}}) &\leftarrow \psi(t_{\text{obs}}) \vee (\text{data}_{\text{referral}} \neq \text{Home} \wedge \text{data}_{\text{state}} = \text{Unstable}) \\ &= (\text{void} \vee (\text{data}_{\text{referral}} \neq \text{Home} \wedge \text{data}_{\text{state}} = \text{Unstable})) \\ &= (\text{data}_{\text{referral}} \neq \text{Home} \wedge \text{data}_{\text{state}} = \text{Unstable}). \end{aligned}$$

Thus, we discovered the partly overlapping guards of the hospital process shown in Figure 10.1: (1) stable patients with a referral to the ward or a tertiary hospital may also be discharged instead of being transferred; and (2) unstable patients with a referral to the ward or a tertiary hospital may also be observed instead of being transferred.

10.3.4 Dealing With Real-life Event Logs

In Section 10.3.1 made several assumptions to explain the key idea: the event log fits the Petri net perfectly, the set of attributes written events is consistent throughout the log, and every attribute value is initialized in the first event. Generally, real-life event logs do not satisfy these requirements. We already discussed how to deal with the assumption of a perfectly fitting event log. Next, we show how to deal with the remaining two assumptions: attributes are recorded consistently and initial values are available.

Inconsistent Attributes. We restricted the set of write operations for a transition to those variables that are consistently given a value by every event connected to this transition. In real-life event logs attribute values can be missing due to temporary recording errors leading to an inconsistent recording of attributes for a given transition. Moreover, the alignment might need to introduce artificial events in case events are missing. For these events, the attribute values are unknown. Therefore, we introduce a user-defined threshold K like in [LA13b] and *add a way to deal with missing values* to it. A variable $v \in V$ is added to the set of write operations of a transition $t \in T$ when the variable is observed to be given a value by $K\%$ of the events e of transition t (i. e., $trans(e) = t$). As a result, attributes might be missing from the set of attributes written for an event $e \in E$, i.e., $dom(\#(e)) \neq wr(trans(e))$. Every time an event e does not assign a value to a variable v even though it should (i. e., $v \in wr(trans(e))$ and $v \notin dom(\#(e))$), we assume $latest(e)(v)$ to be \diamond . Symbol \diamond indicates that the value is *missing*. Decision tree building algorithms, such as C4.5, can deal with such missing values.

Unassigned Attributes. Classical decision trees cannot deal with uninitialized attributes (similar to NULL values in databases). In real-life event logs, attributes might be uninitialized if some of the first events of the log's traces do not assign a value to all attributes. This issue can be mitigated by defining default values that are used when attribute have not taken on values yet. For example, for an attribute approval that is observed to be assigned values {0, 1} in the event log we may use -1 as default value since it is not used in the real execution data. Hence, rules based on the default value can be discovered, e. g., reject claims with uninitialized approval < 0.

10.4 Evaluation

We evaluated our technique using two real-life data sets, and compare the obtained results to standard methods like decision tree induction algorithms. An implementation of our technique is available in the *MultiPerspectiveExplorer* [MLR15b]

package of the open-source process mining framework ProM 6.7, which will be presented in Section 11.2.

10.4.1 Evaluation Setup

Performance Measures. We measured the performance of the approaches in terms of a local fitness measure and a local precision measure. We use measures local to a specific decision point since we are only interested in how the approaches trade-off precision and fitness on that decision point. The local precision measure is described in Section 6.3. We measure the local fitness of a decision-point place $p \in P$ based on how often guard expressions on output transitions of the place are violated in an alignment of the event log to the process model. Let $E_p \subseteq E$ be the set of events that are aligned to output transitions $p\bullet$ of place p .

$$\text{localFitness}(\text{LTS}, L, p) = 1 - \frac{|\{e \in E_p \mid \text{Guard associated with } \text{trans}(e) \text{ is violated}\}|}{|E_p|}$$

Approaches. We compared the performance of our approach with three other methods. We chose two methods at the extreme ends of the respective measure, and one method that naïvely introduces overlap. In total, we compared the following four approaches:

- WO.** The model without rules, i. e., the guard `true` is used for all transitions. This results in a perfect place fitness, no guard is violated.
- DTF.** The model with rules discovered by a decision tree as in work [LA13b] using `false` as guard for transitions that are not part of the decision tree. This method will always result in a perfect place precision as there is never more than one enabled transition.
- DTT.** The model with rules the same rules as in DTF, but using `true` as guard for transitions not part of the decision tree. This method naïvely introduces overlap by enabling all these transitions.
- DTO.** The model with rules discovered by our approach as described in Section 10.3.3.

The DTF and WO methods are at the extreme ends of the respective measures. Our approach aims at providing better place fitness (i. e., less violated guards) at the expense of some place precision (i. e., multiple enabled transitions). Therefore, our approach should provide better place fitness than the DTF method together with better place precision than the a model without rules (WO). Method DTT is included to investigate whether our approach improves over a naïve method to introduce overlap.

Event Logs and Process Models. We used two anonymized real-life data sets: *road fines* and *sepsis*. The road fines event log was taken from an information system handling road-traffic fines by an Italian local police force [LM15]. The road fines log contains more than 150,000 cases with approximately 500,000 events. There are 9 data attributes recorded including the fine amount and the payment amount. The sepsis event log [Man16a] contains events about the pathways of patients within a hospital that have a suspicion for sepsis, a life threatening condition typically caused by an infection. This event log contains data recorded during a full year resulting in 1056 cases with about 15,000 events³⁹ and 39 data attributes, e.g., the results of blood tests and information about the patient from checklists filled in by nurses.

For both event logs, we obtained a normative process model of the control-flow and, thus, without any guards. We used the control-flow of the process model presented in [Man+16a] for the road fines data set. For the sepsis data set, we created a model with help of domain experts from the hospital similar to the one presented in [MB17]. Both models allow for most the behavior observed in the logs, but are lacking precision. We checked this using the fitness measure defined for DPNs in Chapter 5 (road fines: 99.7%, sepsis: 98.6%) and the precision measure for DPNs proposed in Chapter 6 (road fines: 63.9%, sepsis: 16.5%). Therefore, both models are good candidates for adding precision through discovered rules.

Experimental Design. We performed experiments for every decision point for the road-fines and sepsis models, with the exception of four decision points of the sepsis model for which no technique was able to discover rules. We used the C4.5 implementation of the WEKA toolkit with activated pruning and a merge ratio mr of 0.05 in all experiments. For each technique, we used 10 different values of *minimum number of instances* (mi) parameter that were equally distributed within a certain interval, which is determined as follows. The smallest value of the interval was chosen such that the discovered guards were not composed by more than 7 terms (cf., Definition 2.8). This choice was based on the assumption that guards with more than 7 terms are too complex and, hence, of no business value. The upper bound of the interval was the smallest value that could still return a rule. In fact, for a too large value of the mi -parameter no rule would be returned. It is worth observing that the interval may have changed with varying the decision point and the technique (DTO, DTT, and DTF).

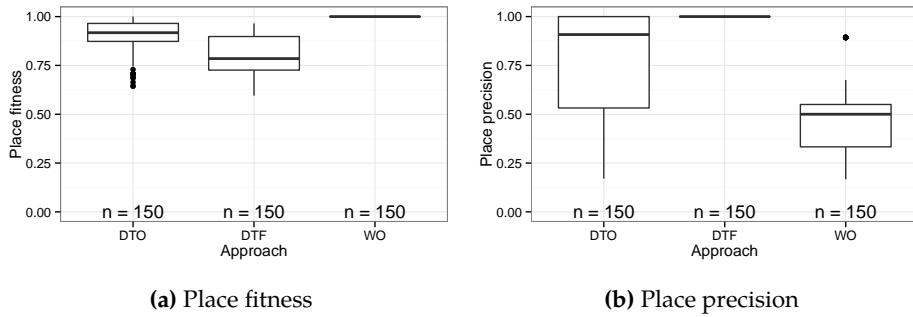


Figure 10.6: Place fitness and local precision achieved by the proposed method (DTO) compared to the standard decision tree classifier (DTF), and the model without guards (WO).

10.4.2 Results and Discussion

We conducted the experiments and recorded the obtained place fitness and place precision for 15 places and 10 parameter settings.⁴⁰ The boxplot in Figure 10.6 shows the average place fitness and place precision achieved by our method and the approaches DTF and WO for all 15 places. We did not include the naïvely approach DTT in the general comparison of fitness and precision as shown in Figure 10.6 since the DTT approach differs from the DTF approach only if there are decision points with more than two outgoing transitions. Only in those cases some transitions may not have been part of the decision tree. Therefore, we compare our method with the naive DTT approach for all decision points with more than two outgoing transitions separately in Figure 10.7.

In the following paragraphs, we interpret the results and discuss the differences between our method and the three other approaches DTT, DTF, and WO.

DTO vs. WO. Compared to WO, the results from our experiment (Figure 10.6b) show clearly that DTO provides rules that increase the place precision against the process model without guards. For some decision points and some parameter settings, our approach deliberately trades precision to obtain better fitting guards. This result is in line with the expectation that our approach returns overlapping rules that lose some precision for a better fitness. The large spread of the obtained place precision indicates that in some cases our method discovers rules with too much overlap, i.e., the discovered overlapping rule is not much more precise than

³⁹Please note that we used a variant of the sepsis cases event log that contains more data than the one made publicly available as [Man16a]. Due to privacy concerns, we cannot disclose that event log.

⁴⁰The data used for the evaluation is available under <http://purl.tue.nl/844997340832257>. For confidentiality reasons we cannot share the variant of the sepsis event log that was used.

no rule at all.

DTO vs. DTF. The experimental results show that DTO discovers decision rules that lead to a better place fitness than the rules discovered by DTF (Figure 10.6a) with, on average, a limited trade-off for lower precision. The outliers for DTO in Figure 10.6a deserve some discussion. We inspected them and found that for some combinations of parameter settings and places, our approach failed to discover overlapping guards. It discovers the exact same rules as returned by DTT. Mostly, this happens for decision points with only two outgoing transitions and high settings of the mi parameter. This can be expected for decision trees with instances from two classes {A, B}. The wrong instances on a leaf $l = (expr, A)$, which predicts transition A, can only belong to the other transition B. Therefore, our approach will not discover a second decision tree with leafs predicting B, but rather use rule $expr$ from leaf l for the majority vote transition B. Our approach only allows this if the number of instances for B is above the setting of the mi parameter. Thus, for high settings of mi and decision points with two outgoing transitions our approach is unlikely to improve over the normal decision tree classifier approach.

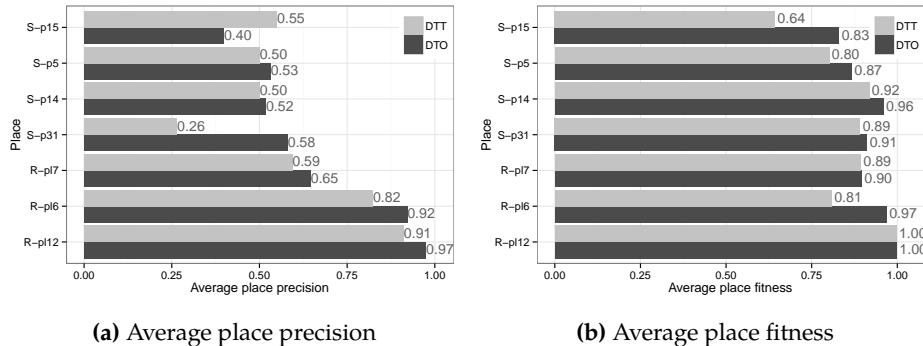


Figure 10.7: Average place fitness and place precision achieved by the DTO method compared to the DTT method. Here, only decision points with more than two outgoing transitions are shown.

DTO vs. DTT. We also compared DTO against DTT, which naïvely assumes the guard true for transitions that are not predicted by the decision tree. For this comparison, we compared the results for decision points with more than two outgoing transitions, $|p\bullet| > 2$, as the results obtained through DTT can only differ from the results of DTF only for these decision points. Figure 10.7 shows the fitness and precision for those places averaged over the considered 10 parameter settings. Each place is given a name for reference. The results in Figure 10.7a show that

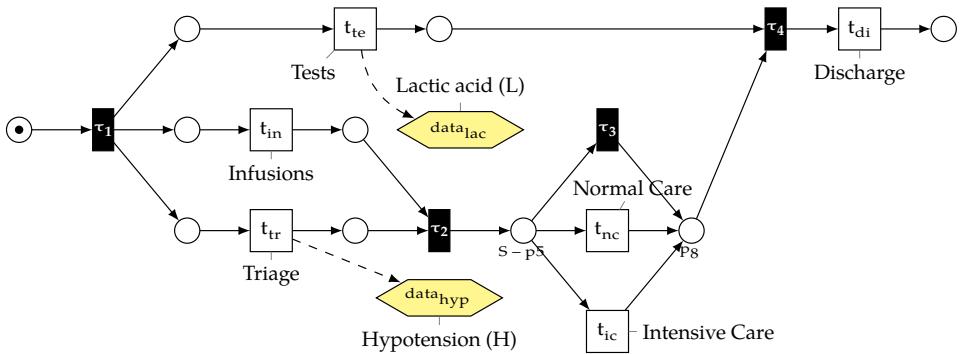


Figure 10.8: Simplified variant of the process model used for evaluating the decision mining method on the sepsis cases event log. After a triage form is filled, infusions are given and blood tests are taken patients are admitted to normal care, intensive care, or not admitted (τ_3). Two relevant attributes are recorded: lactic acid, (L) and hypotension (H).

DTO is able to discover overlapping guards that fit the observations better: for all of the considered decision points, the decision rules returned by our approach increase the place fitness against those rules returned by DTT. Furthermore, the results show that for all except one decision point our approach discovered rules with higher place precision. In other words, it discovered more precise guards without loosing fitness. In fact, for decision point S-p31 our approach obtained an average place precision of 0.58 whereas the rule returned by the DTT approach scores only 0.26. Our approach discovered guards for all six outgoing transitions whereas DTT only discovers guards for three transitions. On the remaining three DTT used true as a guard, i.e., always enabled. The only decision point for which DTO obtained a worse precision score than DTT is S-p15. Our approach discovered guards that correspond to true for all three alternatives. This is possible since our method discovered overlapping rules in the form of $x = \text{true} \vee x = \text{false}$ and we employ a rudimentary tautology check, which reduces such expression to the guard true. However, this is not necessarily a bad representation of the observed data. In fact, the guards discovered by DTT cause the lowest place fitness in our experiment 0.64, i.e., the guards discovered by DTT are wrong in one third of the cases.

Example. Figure 10.8 shows a part of the DPN that we used for the sepsis data set. Table 10.3 shows the guards discovered by DTF, DTT and DTO, the approach presented in this chapter, for the three alternative activities on decision point S-p5. All rules are based on two attributes: Lactic acid (L) and Hypotension (H). DTF discovers the rule that patients with a lactic acid measurement (i.e., $L > 0$) are generally admitted to normal care and patients without lactic acid measurement

Table 10.3: Guards discovered by the compared approaches at decision point S-p5

Approach	Normal Care	Intensive Care	Not Admitted (τ_3)
DTF	$L > 0$	false	$L \leq 0$
DTT	$L > 0$	true	$L \leq 0$
DTO	$L > 0$	$L > 0 \wedge H = \text{true}$	$(L > 0 \wedge H = \text{false}) \vee L \leq 0$

$(L \leq 0)$ leave the hospital. The guard for the admission to intensive care is returned as false. This leads to the situation where patients are never admitted to intensive care even if it is part of the model and observed. Obviously, this cannot be correct. DTF is unable to find a mutually-exclusive rule that includes this alternative activity given the information recorded in the event log. DTT discovers the same rules but naively assumes the guard true for admission to intensive care. Clearly, the DTT results are not satisfying as DTT would convey no rules about the admission of patients to intensive care. Our approach – DTO – discovers that patients with a lactic acid measurement ($L > 0$) can always be admitted to normal care. As an alternative to normal care, for case in which $H = \text{true}$ holds (i. e., the hypotension flag was checked) patients can also be admitted to intensive care; otherwise, if $H = \text{false}$ (i. e., the hypotension flag was not checked) patients leave the hospital. The guards for the activities overlap and the final decision is likely to be made on the basis of contextual factors, which are not encoded in the event log.

10.5 Related Work

We discuss the related work in the fields of *process mining*, *classification*, and *rule learning*. Moreover, we also discuss the relation to the *DMN standard* and related research.

Process mining. There are several approaches for decision mining in the context of process mining [AAD11; BBW16; BW16; Cat+14; Dun+14; Gri+04; GSP14; JJ13; LA13b; LDG13; RA06]. All these methods assume to be given an event log with historical data about the process. Furthermore, in all of these approaches the decision-mining problem is translated into a classification problem, and solved using classification analysis techniques such as C4.5 [Qui93]. All of these approaches discover only mutually-exclusive rules, i. e., there is no other work about discovering overlapping rules from data labeled with single classes for decision mining in the context of process mining.

Classification. Most related from the traditional classification field to our approach is work about *multi-label classification* [Bou+04; TK07]. In a multi-label classification

problem classes are not mutually exclusive, instances can be labeled with multiple classes, and the goal is to find the correct set of classes for unseen instances. Our setting is still different as we deal with instances that are only associated with one class, i. e., the executed transition. *Classifier chains methods* are the closest to our work. They decompose the problem into multiple binary classification problems, one for each label [Rea+11]. These methods also assume that instances are labeled with multiple classes, which is not the case in our setting.

Rule learning. Methods for association rule mining [AIS93], rule learning based on rough sets [Ste98], and rule learning based on fuzzy sets [Zad65] are also related to our work. Association rule mining methods return a large set of, potentially overlapping, rules. Methods based on rough sets are based on the idea that given the information in the observation instances (or a subset of attributes of the instances) classes may be indiscernible, e. g., given two classes A and B and instances $[(X \rightarrow 1, Y \rightarrow 0), A), ((X \rightarrow 1, Y \rightarrow 1), B)]$ it is not possible to define a rule to distinguish between class A and class B based on attribute X. Rough set based rule learning method can discover *certain rules* (i. e., classes can be distinguished) and *approximate rules* (i. e., the rules overlap) [Ste98]. Discovered approximate rules would be suitable candidates for overlapping guard expressions. However, the main problem of these rule learning methods is that a potentially large set of rules is usually returned, failing to provide insights that are easy to interpret. As an alternative to our approach, rule learners based on fuzzy sets, e. g., fuzzy decision trees [Jan98], could also be used to obtain rules that may overlap due to the non-binary membership function employed in fuzzy sets. Appropriate membership functions would need to be inferred (e. g., by using an approach such as NEFCLASS [Nau03]) or designed manually. Moreover, the precision and fitness measures for DPNs would need to be adapted for fuzzy rules.

DMN standard and Decision Tables. DMN is a standard for the modeling of decisions using decision requirements diagrams and decision tables. Decision requirement diagrams are envisioned as a notation for the internal structure of decision marking. Decision tables are used for the decision logic [DMN16]. DMN takes a broader view at decision making. It is not only concerned with routing decisions that describe which activities may be executed next. Moreover, DMN also considers forms of decision that are different from routing decision in processes, e. g., DMN may be used to model the decision what percentage of discount is granted to a customer. Research related to DMN has also been addressing the discovery and management of decision rules. In work [Cal+16] DMN decision tables are analyzed and overlapping rules are detected. This approach is useful when overlap between rules is considered as quality problem. As motivated in Section 10.3.3, there are situations in which overlapping rules are inevitable because of the non-deterministic nature of human behavior. Amongst the research on DMN, work on fuzzy decision

tables [Baz+17; VWC96] shares a similar goal: representing imprecise decision rules. In [VWC96] decision tables based on fuzzy rules are introduced and their semantics are discussed. However, the decision table is not automatically discovered. In work [Baz+17], fuzzy logic is incorporated into DMN decision tables and fuzzy rule learners are employed to infer fuzzy classification rules from event data. The result is a set of fuzzy rules that can be transformed to natural language statements like: “If loan duration is long, then risk is very high” [Baz+17]. Differently from our approach, the precision and fitness of the obtained process model is not considered and overlap between rules is introduced only for numerical attributes. Moreover, the execution semantics proposed for fuzzy decision tables selects a single rule to be executed at runtime, i. e., no overlap is introduced at runtime, which is different from our goal. Compared to our approach, DMN is not only concerned with the decision logic. It also models the dependencies on data that is required to yield a decision: the decision requirements diagram. There is recent work [De +17b] that aims to automatically extract a decision requirements diagram from an event log. The write operations preceding a decision point in a DPN also make the required data for the decision explicit. Still, the structure of a decision requirement diagram cannot be directly inferred from the write operations of a DPN.

10.6 Conclusion

We proposed a new technique for the discovery of overlapping rules in process models using event data. Existing techniques only return rules that assume completely deterministic decisions. This assumption often does not hold in real-life settings. The data recorded in event log is often incomplete and real-life decision making may be non-deterministic according to this information.

10.6.1 Contribution

Our technique is the *first proposal* of a discovery technique for decision logic in process models that introduces overlapping rules. This chapter is based on our publication [Man+16b]. The technique aims to create process models that trade the precision of mutually-exclusive rules for the fitness of overlapping rules when the observed behavior gives evidence to such rules. To evaluate our technique we used several real-life data sets. We measured the fitness and precision of the process models with discovered rules (cf., the conformance checking technique in Part II). The evaluation showed that our technique is able to produce models with overlapping rules. The discovered models fit the observed behavior better without loosing too much precision. For some decision points with more than 2 alternative activities, our technique returns rules that are both more fitting and more precise than the existing method [LA13b].

10.6.2 Limitations

The results show that our technique is successful in uncovering overlapping rules in processes from event logs, and that these rules provide in some cases a much better characterization of the observed behavior. Still, the proposed technique has some limitations.

- We evaluated our technique to provide good results on two real-life event logs. More experiments using event logs from different settings are required. Moreover, a qualitative evaluation that involves domain experts to judge the pragmatic quality of the discovered rules would be required.
- An inherent limitation of our approach is that it only uses the majority vote to introduce overlapping guards for a decision point with two output transitions. This might cause the guard of one transition to be turned into the rule true, e.g., when the initial guards were based on a single condition. It might also affect the place fitness negatively in case no initial condition was found for the transition.
- Our approach tends to discover guards that are more complex: Guard may become unreadable if algorithm's parameters are not carefully chosen. In the implementation of our method we attempt to simplify rules by removing tautologies (e.g., rule $X = \text{true} \vee X = \text{false}$ is simplified to true) and collapsing numeric intervals (e.g., rule $X > 10 \wedge X > 20$ is simplified to $X > 20$). Simplification of boolean expressions is not trivial in the general case; however, more advanced methods from this domain could be used.

10.6.3 Future Work

There are several directions for future work that are worthwhile exploring.

- It would be beneficial to investigate the application of other techniques different from decision trees to decision mining and decision mining of overlapping rules. It would be useful to have a parameter that influences the expected degree of overlap of the rules to be found. This would allow us to steer the discovery of data-aware process models in the spectrum between fully fitting and fully precision models. A possible realization could be based on rule induction approaches, e.g., based on association rules [AIS93] or rough sets [Ste98]. Rule mining approaches based on rough sets can yield overlapping rules that can be steered in the spectrum from mutually-exclusive (crisp part of the set) towards overlapping (rough part of the set). However, the challenge on how to select appropriate rules from a large set of candidates needs to be addressed.

- Another important future work is to address limitations of decision mining techniques for data sets with imbalanced distributions of classes. Imbalanced distributions are a phenomenon often found in business process (e.g., decisions regarding exceptions or infrequently visited paths of the model). Although we note that our technique is able to reveal rules when one transition is only observed for a small fraction of the cases, a more thorough investigation of this phenomenon is needed.
- Linking our method with work [Baz+17; Cal+16; De +17b] that has been done on discovering decision requirement diagrams and decision tables such as defined by the DMN standard would be interesting. Our method could benefit from integrating decision requirement diagrams into DPN and using decision tables as an alternative representation for guard-based decision rules. Vice versa, overlapping decision rules could be integrated into DMN.

Part IV

Applications

Chapter 11 We present two interactive tools that implement our proposed methods in the process mining framework ProM.

Chapters 12 to 15 We describe four case studies that we conducted using the method proposed in this thesis.

11 Tool Support

In the following two sections, we describe two tools that were developed as part of this thesis. We start in Section 11.1 by presenting the *Interactive Data-aware Heuristic Miner (iDHM)*, which implements the process discovery technique described in Chapter 8 and additionally constitutes a general platform for heuristic process discovery. Then, in Section 11.2 we describe the *Multi-perspective Process Explorer (MPE)*, which realizes the methods described in Chapters 5, 6 and 10.

11.1 Interactive Data-aware Heuristic Miner

First, we present the *Interactive Data-aware Heuristics Miner (iDHM)*, an open-source tool in the ProM framework⁴¹ that implements the process discovery method presented in Chapter 8. Parts of the content of this section were published in [MLR17]. The iDHM aims to address the following five shortcomings of many of the existing heuristic-based process-discovery tools:

1. the large parameter space of heuristic methods needs to be explored manually;
2. several of the many available heuristics can be chosen from;
3. data attributes (i.e., the event payload) are not used for process discovery;
4. discovered C-Nets are not visualized as described in literature; and
5. existing tools do not give reliable quality diagnostics for the discovered models despite the lack of quality guarantees offered by heuristic approaches.

The iDHM provides *interactive exploration* of the parameter space and includes built-in conformance checking to diagnose the quality of the discovered model. Thus, it is easier to *explore a large parameter space* and directly *spot conformance issues*, e.g., deviating and missing behavior in the event log. Furthermore, the iDHM uses data attributes of the event log to reveal infrequent conditional dependencies as described in our data-aware heuristic process discovery method in Chapter 8.

The iDHM visualizes discovered models as Causal Nets (C-Nets) (cf., Section 3.3) and Data Causal Nets (DC-Nets) (cf., Section 8.4). C-Nets depict the split- and join gateways of a process in a concise graphical notation with *clear semantics*. Existing tools do not directly visualize the C-Net notation but rely on a tabular or external description of the discovered split- and join gateways. The *exact semantics are not*

⁴¹Available in the *DataAwareCNetMiner* package of ProM 6.7: <http://promtools.org>

visible directly in those tools. Finally, the iDHM is build with a plug-in architecture that allows to add new heuristics.

11.1.1 Overview of the iDHM

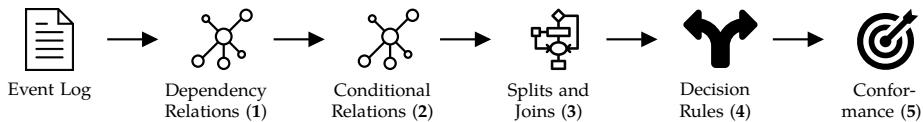


Figure 11.1: An overview of the five discovery steps of the iDHM.

We distinguish five steps of data-aware process discovery as depicted in Figure 11.1:

1. mining dependencies,
2. mining conditional dependencies,
3. mining split- and join information,
4. discovering decision rules, and
5. checking conformance.

11.1.2 Walk-through of the iDHM

We present a walk-through of the iDHM using an event log obtained from the billing process of a regional hospital in The Netherlands. A more comprehensive analysis of this process is described in Chapter 15. Figure 11.2 shows the main screen of the iDHM. The required input is an event log in the XES format. Traces from the event log may be filtered by using a SQL-like query syntax ①. The figures in this section show the output in form of a DC-Net, but other output formats such as Petri net or Data Petri net can also be chosen based on the translation described in Section 8.4 ②. Each step of the iDHM can be configured ③ and several thresholds (cf., θ_{obs} , θ_{dep} , θ_{bin} , and θ_{con} defined in Section 8.3.3) may be used to configure the employed heuristics ④. At the bottom of the screen a legend explains the color coding that is used ⑤.

Step 1: Discover Dependency Relations

First, the set of dependency relations is determined. A dependency relation (a, b) between two activities represents a causal dependency from activity a to activity b. Figure 11.3 depicts the dependency relations discovered for the threshold settings of the iDHM chosen in Figure 11.2. Figure 11.3 is automatically generated by the iDHM, i. e., it is possible to view the results of each step in isolation. For example, activity DELETE depends on a prior execution of activity NEW. This is depicted as a directed edge between both activities. Only strong dependencies that

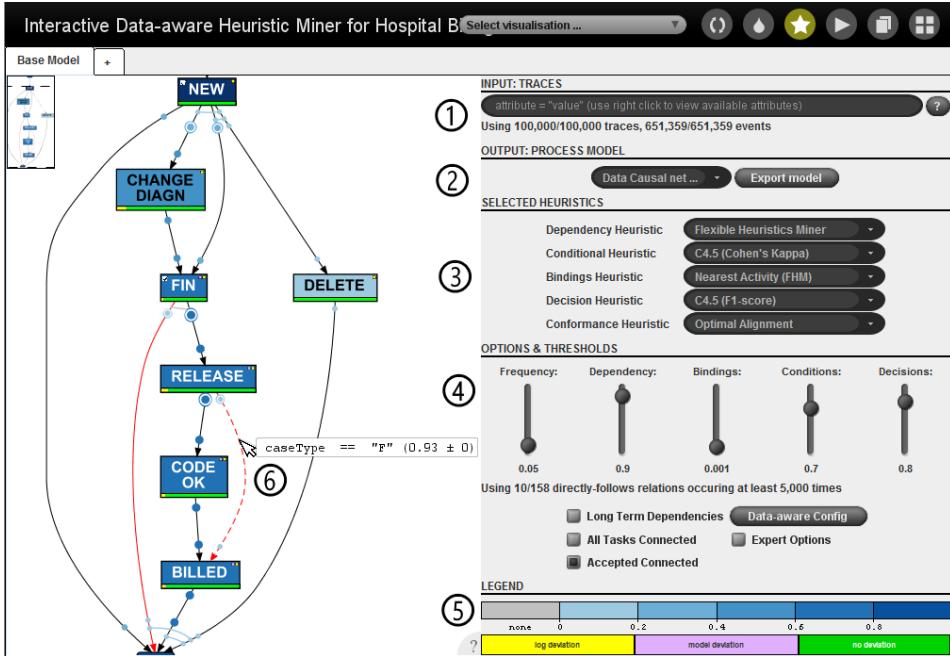


Figure 11.2: The main screen of the iDHM shows conformance information projected on a discovered DC-Net. The dotted edge between activity RELEASE and activity BILLED is currently selected and the discovered dependency condition is shown.

exceed a configured *dependency threshold* and are observed more often than a configured observation *frequency threshold* are included. The configuration corresponds to setting the thresholds described in Section 8.3.3 to $\theta_{obs} = 0.05$ and $\theta_{dep} = 0.9$. Several heuristics have been proposed to discover dependency relations and their strength. A suitable method can be chosen from four implemented methods: the Flexible Heuristics Miner (FHM) [WR11], the Alpha Miner [AWM04], the Fuzzy Miner [GA07], and the average of these miners. In Figure 11.2, we used the FHM method. New methods can be added as plug-ins to the iDHM.

Step 2: Discover Conditional Dependency Relations

Second, the data perspective is taken into account when discovering the control flow of a process. Classification techniques are used to reveal data dependencies between activities. These data dependencies are used to distinguish random noise from infrequent conditional dependencies. This greatly extends existing techniques, which are solely based on the control-flow perspective and, hence, would disregard such infrequent behavior as noise. *Conditional dependencies* may provide insights for

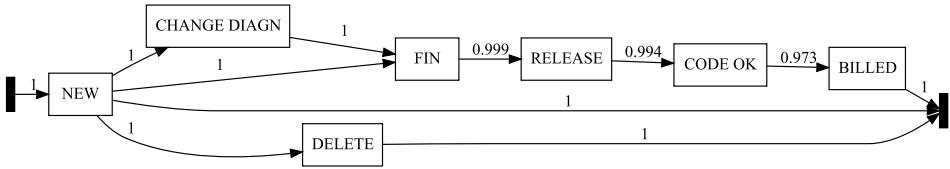


Figure 11.3: Dependency relations determined by the iDHM in its first step.

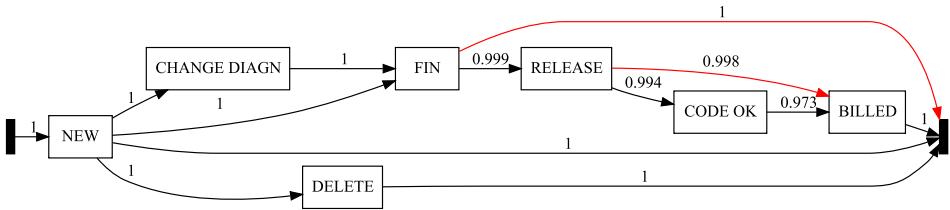


Figure 11.4: Two conditional dependencies highlighted in red, were added in the second step.

process analysts since they could indicate, e.g., workarounds and deviations from normal process behavior. In Figure 11.4, e.g., the dependency between FIN and the end of the process is a conditional dependency and the dependency between RELEASE and BILLED is also a conditional dependency.

In the resulting C-Net, conditional edges are highlighted in red. Further statistics on the discovered condition can be obtained by right-clicking on the respective edges of the DC-Net (or C-Net), e.g., in Figure 11.2 the edge between FIN and the end of the process is selected and a pop-up (see ⑥) shows that data condition closeCode = H was discovered for that edge, which was associated with a quality score of 0.98. We only include conditional dependencies for which the quality of the underlying data condition exceeds the configured condition threshold (in Figure 11.2 we use $\theta_{con} = 0.7$). The iDHM implements our data-aware heuristic discovery method as presented in Chapter 8 to discover conditional dependencies. We implemented two variants of this technique: a plug-in that evaluates the data conditions with Cohen's kappa as described in Section 8.3.2 and a plug-in that evaluates the data conditions using the standard F1-score for binary classification. In this walk-through, we use Cohen's kappa as quality measure.

Step 3: Discover Split- and Join Information

Third, the split- and join information of the C-Net, i.e., its input- and output bindings, need to be discovered to obtain a model with precise semantics. The bindings of the C-Net are visualized as dots on the edges between two activities. Disconnected dots represent exclusive splits (XOR) and connected dots represent parallel

splits (AND). In Figure 11.5 there are two output bindings defined for activity RELEASE. After executing RELEASE an exclusive choice between either BILLED or CODE OK is modeled, i. e., the two binding dots are not connected with each other. Currently, the heuristic proposed by the FHM in [WR11] is implemented and only frequent bindings exceeding the *bindings threshold* are displayed. In future work, other heuristics, e.g., the structuring approach presented in [Aug+16] may be added.

Step 4: Discover Decision Rules

Fourth, decision rules that determine which of the possible output bindings may be activated are discovered. Bindings with attached decision rules, i.e., guarded bindings, are depicted by a double border. In Figure 11.5 decision rules could be discovered both output bindings of RELEASE. The binding from RELEASE to CODE OK is activated only for cases with the *caseType* attribute values A, B, and D. The binding from RELEASE to BILLED is activated only for *caseType* attribute values C, F, and G. Additional information on the decision rule such as details on its evaluation in terms of the chosen quality measure can be accessed by right-clicking on the binding. Decision rules may be filtered based on a decision-rule quality threshold. We implemented two decision mining methods, one based on C4.5 decision trees and one based on overlapping decision rules described in Chapter 10.

Step 5: Check Conformance of the Model

Finally, we apply conformance checking techniques on the C-Net and the event log to project frequencies on the activities and bindings. In Figure 11.5, the size and color of the binding dots scales with the frequency of their activation estimated by the employed conformance technique. For example, the binding from RELEASE to CODE OK occurs much more often than the other output bindings of RELEASE. Activities are also color-coded, cf. the different colors of the activities in Figure 11.2. Moreover, conformance problems are projected on the activities. Here, we use the conversion between DC-Nets and DPN that is de-

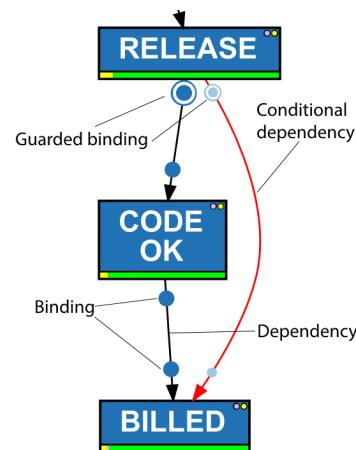


Figure 11.5: Bindings and guarded bindings.

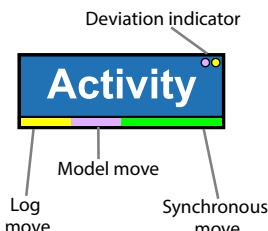


Figure 11.6: Conformance statistics

scribed in Section 8.4 and use the multi-perspective balanced alignment method presented in Chapter 5 to compute an salignment (cf., Section 4.3). Following the definition of the alignment, we categorize deviations in:

- *log moves*, i. e., the event cannot be matched to the process model;
- *model moves*, i. e., the model activity cannot not be matched to an event; and
- *synchronous moves*, for which there is no control-flow discrepancy between event log and model.

Statistics on the number of log-, model-, and correct moves are projected on the activities using the color-coding shown in Figure 11.6. Additionally, we add a small circle of the respective color to the top left of the activity as binary deviation indicator to signal the presence or absence of conformance issues. This helps to enables to spot activities with a low number of conformance issues, which would be difficult using only the color-coded bar below the activity. Moreover, it is possible to convert the DC-Net to a DPN as shown in Figure 11.7 and use it with the Multi-perspective Explorer (MPE) tool (cf., Section 11.2) to further analyze the deviations in more detail and compute the fitness and precision measures proposed in Section 4.4 and Chapter 6. We describe the MPE tool in Section 11.2.

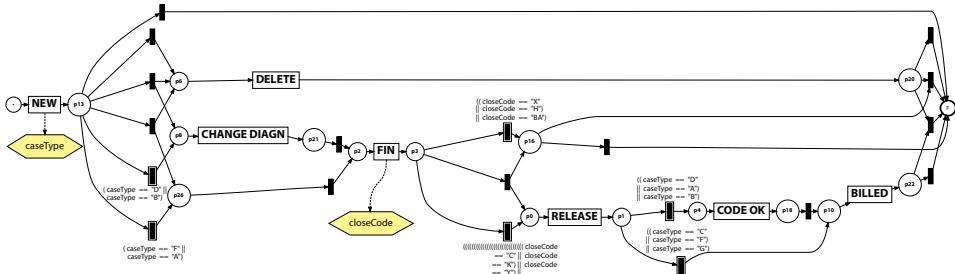


Figure 11.7: DPN converted from the discovered DC-Net by the iDHM tool. The DPN can be used, e. g., for further analysis in the MPE tool.

Several heuristic and exact approaches are available to compute the conformance statistic. Currently, we only implement our multi-perspective balanced alignment method Chapter 5. Possible additions in future work would be, e. g., the heuristic execution semantics proposed in [Bro14], the fuzzy log replay proposed in [Roz10], or approaches that approximate an optimal alignments, e. g., work [Don+17; TC16], to provide faster feedback.

11.1.3 Plug-in Architecture

As described, it is possible to choose from one of the implemented heuristics for each of the steps 1-5. The iDHM is built on a plug-in architecture, each of the heuristics is implemented as plug-in of the iDHM; thus, it is possible to extend

the tool with new heuristics by implementing a Java interface and adding the new class to ProM.⁴²

11.1.4 Conclusion of the iDHM

We presented the iDHM, an interactive data-aware heuristic process discovery tool. The tool integrates several heuristics and can be used as testbed for new heuristics. The tool integrates with the ProM framework so that the obtained results can be used for further analysis. We showed that the iDHM addresses some of the common shortcomings of academic tools for heuristic process discovery. It allows for quick exploration of the parameter space (1) and several different heuristics (2). It uses data attributes for control-flow discovery (3) and visualizes the result as models with clear semantics (4). Finally, it provides built-in conformance checking (5).

We successfully tested the iDHM on several non-trivial real-life event logs of considerable size, which confirms that it has reached a high degree of maturity. In this paper, we showed its application to the event logs of a hospital billing process with 100,000 traces and more than 450,000 events. The tool has also been successfully applied to larger events logs with more than 1,000,000 traces and 6,000,000 events.

A limitation of the iDHM is its dependence on an automatic layout based on the GraphViz software. Very complex C-Nets are unlikely to be readable and it is not yet possible to manually adjust their layout. As future work we plan to apply the iDHM to more case studies and further improve its usability. Moreover, we plan to enrich the C-Net notation with perspectives different than control-flow.

11.2 Multi-perspective Process Explorer

Next, we describe the *Multi-perspective Process Explorer* (MPE), a tool that is tailored towards multi-perspective process exploration for enhancement and conformance analysis. It integrates our work on multi-perspective process mining as described in Chapters 5, 6 and 10 as well as existing decision mining methods [LA13b] in a scalable and flexible tool. Moreover, the MPE provides interactive data-aware visualizations and filtering facilities.

Applying multi-perspective process mining techniques in practice is a laborious task, especially in cases when the data contains a large number of different attributes with high variability. A substantial amount of manual work by analysts is required, because they need to filter and transform event data as well as to select relevant features. Also, results need to be explored and, if not satisfactory, these steps need to be repeated multiple times by hand. The MPE aims to support this task and provides three main features:

⁴²Details on developing iDHM plug-ins can be found at <http://fmannhardt.de/g/dhm>

- integration of multi-perspective *conformance checking*, analysis of deviations and *performance analysis* techniques;
- interactive efficient *discovery of decision rules* governing the process;
- built-in *filtering* and event log *exploration* based on context-sensitive charts and a *trace variants* explorer.

In the following sections, we give an overview on the functionality of the MPE and present a brief walk through showing the sequence of steps required to *discover* and *evaluate* a multi-perspective process model. Parts of the work presented here was published in [MLR15b].

11.2.1 Overview of the MPE

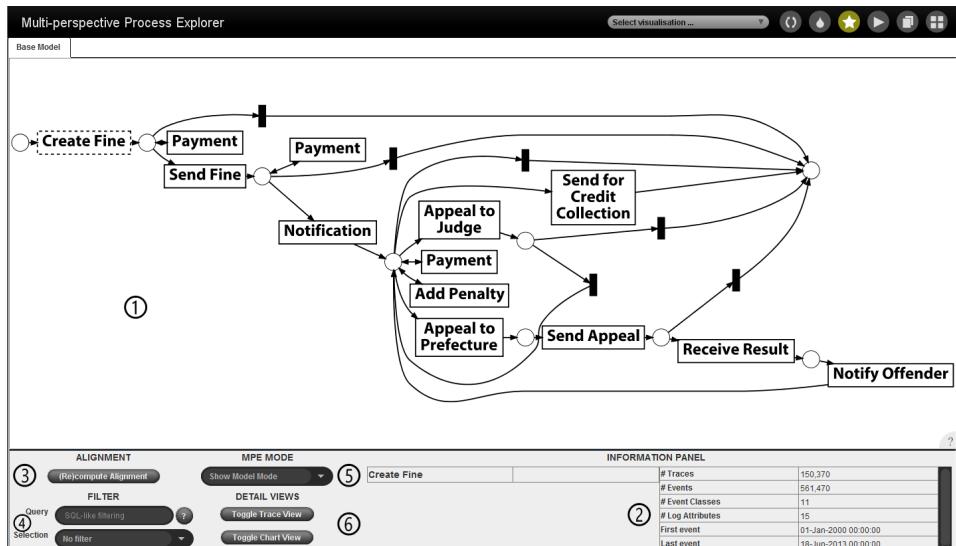


Figure 11.8: Main screen of the MPE showing the input process model (base model).

The MPE is implemented as plug-in of the ProM framework in the *DataAwareExplorer* package⁴³. Starting point for the usage of the MPE is an event log and a process model. Both the event log and the model need to be available as objects in the ProM framework.

Event log. ProM supports loading event logs in the XES format [IEECCIS16] and from CSV files. To support large event logs in which events carry several attributes as payload, we developed the XESLite library [Man16b], which efficiently stores

⁴³The *DataAwareExplorer* is available in ProM 6.7.

typical event logs on commodity hardware. XESLite is used by the MPE to store the input event log as well as the alignment between the event log and the process model. The alignment is stored in a standard format, which extends the XES standard for the storage of alignments.⁴⁴

Process model. The MPE supports process models in the DPN notation. This includes also classical Petri net without variables and guards. Petri nets can be discovered, e. g., by using the many process-discovery techniques available in ProM. Alternatively, ProM supports the import of Petri nets in the PNML format. Thus, process models can be created manually with an editor such as WoPeD⁴⁵ or the ProM plug-ins *Create Petri net (Text language based)*⁴⁶ and *Create DPN (Text language based)*. Optionally, the MPE can also be used with hand-made DPNs that already contains decision rules.

Figure 11.8 shows the start screen of the MPE when applied to a process model (i. e., Petri net) of the road traffic fine management process and the corresponding publicly available event log [LM15]. We use this data set to describe the functionality of the MPE. The event log, the process, and the normative process model are described in more detail as part of the case study in Chapter 12. The user interface of the MPE consists of six parts that are marked with numbers in Figure 11.8:

- ① The main area of the MPE screen shows the process model and, in some modes, additional information is projected on top of the model. Selected nodes (e. g., here *Create Fine*) are highlighted with dashed borders.
- ② In the lower area of the MPE information on the currently selected model elements and general information on the model and the event log is shown.
- ③ Integral part of the MPE is the computation of an alignment between the process model and the event log. The MPE uses the balanced multi-perspective alignment method presented in Chapter 5 to compute such an alignment. The alignment computation can be started and configured using the button *(Re)compute Alignment*.
- ④ Often, it is useful to focus on a specific sub set of the traces based on data attributes stored in the event log. The MPE provides a flexible filtering mechanism, which can be used to query a sub set of traces based on a SQL-like query language. Moreover, it is possible to filter the event log based on selecting model elements, e. g., to retain only those traces in which certain transitions were executed or certain places were marked. This selection based filtering relies on alignments to establish the connection between model elements and log events.
- ⑤ The functionality of the MPE is divided in several modes:
 - **model mode,**

⁴⁴The XES alignment extension is published as ProM package *XESAlignmentExtension*.

⁴⁵Available at <http://woped.org/>

⁴⁶Available in the *DataPetriNet* package of ProM 6.7.

- **data discovery mode,**
- **performance mode,**
- **fitness mode, and**
- **precision mode.**

Each of these modes supports a specific use case of the MPE. We will describe the features of each mode in the walk-through of the MPE.

⑥ Finally, it is possible to open two additional views in separate windows:

- The **trace view** allows to explore the traces of the event log and the computed alignments to the process model in more detail.
- The **chart view** allows to explore the distribution of data values during the process.

11.2.2 Walk-through of the MPE

We now describe a step-wise walk-through of the MPE based on the scenario in which we want to enhance the process model of the road traffic fine management process with decision rules, check the quality of the resulting process model, and analyze some parts of the process in more detail. All figures in this section are directly taken from the MPE. In some cases, we did small adjustments on the layout of the models to improve readability.

Step 1: Analysis of the Input Model.

The first step of the walk-through is to analyze the process model that was provided as input, i. e., the normative model for the road traffic fine management process (Figure 11.9) without guard expressions and variables. We determine which paths of the process model are frequented more often and how well the process model represents the observed process behavior as recorded in the event log.

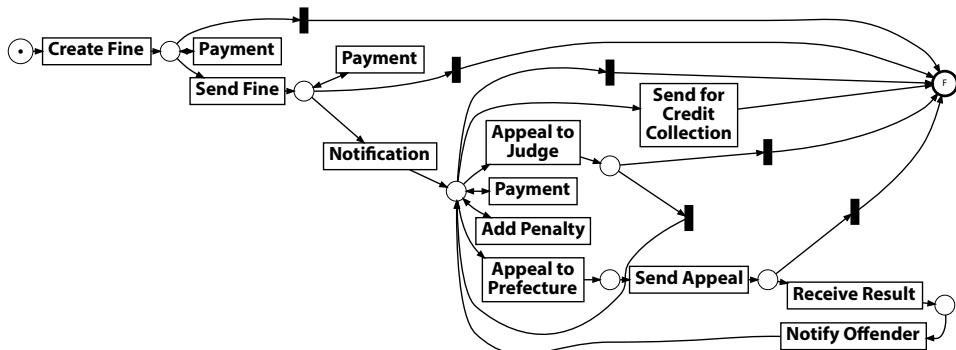


Figure 11.9: Petri net of the road traffic fine management process used as input to the MPE.

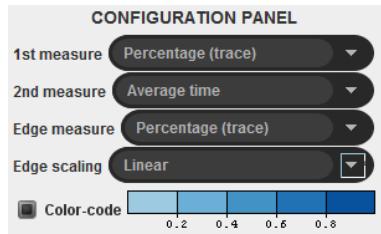


Figure 11.10: Configuration options of the MPE performance mode

First, we first switch the MPE to the **performance** mode. To determine the quality of the model and project frequency information, the MPE computes an optimal alignment between the traces of the event log and the process model. Parameters such as the employed cost function, a mapping between activities and transitions, and a mapping between attributes and variables (cf., the mapping functions λ and v in Section 3.2) can be configured. The default configuration assigns a cost of three for log moves, a cost of three for model moves, and a cost of one for each incorrect write operation. We choose this default setting for the MPE since we assume that the control-flow of the user-defined process model can be trusted more than the guard expressions to be discovered.

Moreover, it is possible to change several *expert configuration options*. We elaborate on two more commonly used options: the choice of the used MILP solver, we support both the open-source solver *lpsolve*⁴⁷ and the commercial solver *Gurobi Optimizer*⁴⁸, and the handling of uninitialized variables. Uninitialized variables may occur in a DPN if a guard expression uses the value of a variable when it is still assigned the initial value (cf., function *in* in Definition 3.6). To simplify some common use cases, we provide two modes to handle the initial value of variables: (1) Uninitialized variables are assumed to be *FREE*, which means that they may take on any value or (2) *NULL* (the standard), which means that they will be mapped to a special value. This may also be used in guard expressions.

After the alignment is computed, it is possible to select which of several available performance measures are projected on the edges and transitions of the DPN. The following measures are supported by the MPE:

- **frequency**, the absolute number of traces traversing an edges;
- **percentage (local)**, the frequency of traces traversing an edge relative to the number of traces reaching or leaving a place of the DPN;
- **percentage (trace)**, the frequency of traces traversing an edge relative to the number of all traces;
- several **waiting time** statistics (average, median, minimum, maximum, first quartile, and third quartile).

⁴⁷<http://lpsolve.sourceforge.net>

⁴⁸<http://www.gurobi.com>

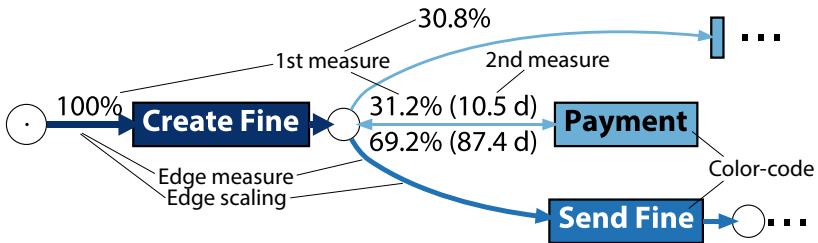


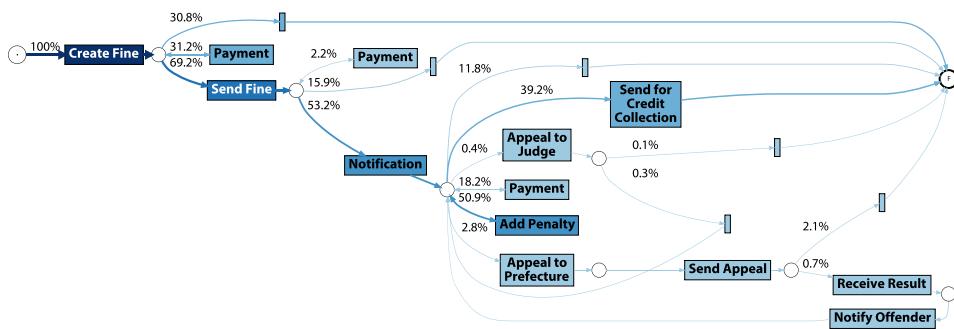
Figure 11.11: Visualization options of the MPE: Two measures can be visualized on the edge labels, the edge width can be scaled according to one measure, and edges and transitions may be colored according to their absolute frequency.

Figure 11.10 shows the configuration screen that allows to change the following options:

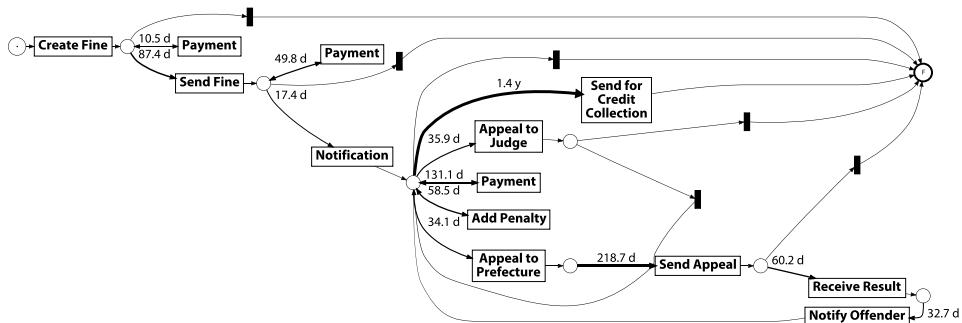
- **1st measure**, configuring the first measure that is shown on the edges;
- **2nd measure**, configuring the second measure that is shown on the edges;
- **edge measure**, configuring the measure that controls the edges width;
- **edge scaling**, configuring the transformation that is applied on the edge measure (linear, square root, logarithm) to prevent outliers from dominating the visible differences; and
- **color-code**, configuring whether edges and transitions should be colored according to their absolute frequency.

The effect of the configuration options chosen in Figure 11.10 on the visualization is shown in Figure 11.11. The overall percentage of traces passing through an edge was selected as first measure and the average waiting time between transition as second measure. Note that measures are only shown when relevant. For example, time-based measures are not shown on edges to routing transitions since the alignment provides no execution time for those. Also, the measures are not shown on the outgoing edges of transitions since the number of traces that passes through the transition is equal to the number of traces that pass through each outgoing edges. By doing so, the visualization is kept compact.

Figure 11.12a shows the visualization that is shown when selected the measure *percentage (trace)* to be projected both on the edge labels (i.e., *1st measure*) and as *edge measure*. Moreover, the transitions and edges are color-coded according to their overall frequency. It is clearly visible that some parts of the process, e.g., *Appeal to Judge* and *Insert Data Appeal to Prefecture*, are executed less often. Figure 11.12b shows the result of projecting the *average waiting time* measure on both the edge labels and the edge width. In Figure 11.12b, we used the *square root* transformation to reduce the visual impact of outliers, since the average waiting time for activity *Send for Credit Collection* is very high (1.4 years) in comparison to the waiting time of the other activities. Note that there is no waiting time for invisible routing transitions



(a) Percentage (trace) measure projected on the process model and color-coded activities.



(b) Average waiting time between enabling of transitions until their execution.

Figure 11.12: Two examples of performance statistics projected on the process model.

since they are not matched to events in the event log.

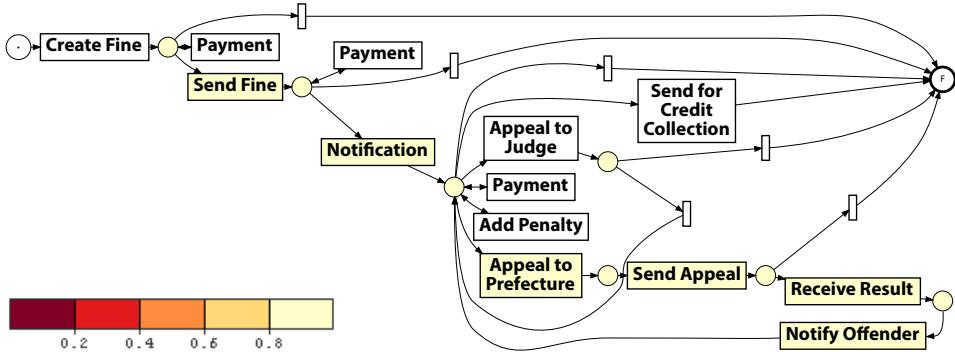


Figure 11.13: Fitness diagnostics projected on the process model.

To analyze how well the input process model represents the observed process behavior, we switch the MPE to the **fitness mode**. Figure 11.13 shows the fitness diagnostics projected on the process model along with the used color scale. The color of a transition t represents a local fitness statistic that is determined based on the number of *model moves* (countModelMoves_t) and *incorrect moves* ($\text{countIncorrectMoves}_t$) observed in the alignments moves of transition t compared to the *overall number of alignment moves* (countAllMoves_t) observed for that transition, i. e., the measure is calculated as:

$$1 - \frac{\text{countModelMoves}_t + \text{countIncorrectMoves}_t}{\text{countAllMoves}_t}.$$

Log moves are not counted towards the local fitness statistics of transitions since they cannot reliably be matched to a certain transition. Therefore, we also introduce a local fitness statistic for places. The local fitness statistic for a place p is based on the *number of log moves observed* while place p was marked with at least one token (countLogMoves_p) compared to the overall number of alignment moves observed while place p contained at least one token (countAllMoves_p), i. e.:

$$1 - \frac{\text{countLogMoves}_p}{\text{countAllMoves}_p}.$$

Most deviations are recorded for transitions *Insert Date Appeal to Prefecture*, *Send Appeal to Prefecture*, *Receive Result Appeal from Prefecture*, and *Notify Result Appeal to Prefecture*. However, none of the deviations are assigned a fitness score less than 0.8.

For the road traffic fine management process, we obtain a good overall fitness-score of 0.996. Thus, the process model is able to describe almost all of the behavior observed in the event log (cf., Section 4.4 for the definition of the fitness-score).

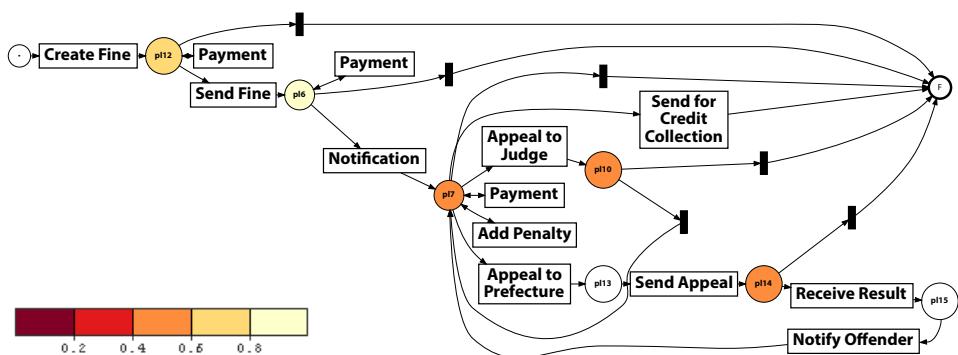


Figure 11.14: Precision diagnostics projected on the process model.

Next to the fitness score, we are also interested in the precision of the process model with regard to the event log. Therefore, we switch the MPE to the *precision mode* and determine the multi-perspective activity precision measure as described in Chapter 6. It is possible to set the set of data attributes that should be considered for the calculation of precision measure (i.e., those attributes are added as variables to the DPN). In our case, we specified the following data attributes of the road traffic fine management event log: *amount*, *dismissal*, *expense*, *lastSent*, *notificationType*, *paymentAmount*, *points*, and *totalPaymentAmount*. The average activity precision score of the road traffic fine management process model is 0.664, which can be seen on the information panel of the MPE (not shown in Figure 11.14). Figure 11.14 depicts the precision diagnostics projected on the process model by the MPE. It shows that places pl7, pl10, pl12, and pl14 are the main source of imprecision.

Step 2: Data-aware Discovery.

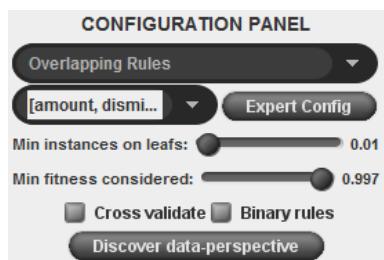
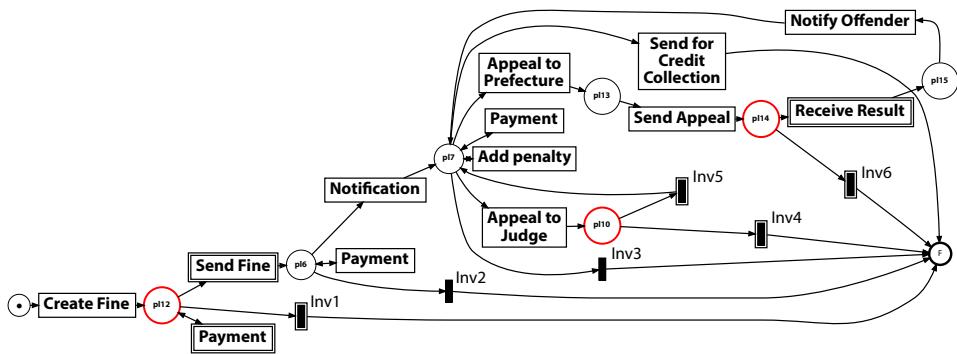


Figure 11.15: Configuration options of the MPE data discovery mode.

In the second step, we discovered decision rules (i.e., the guard function of the DPN) based on the event log to increase the precision of the process model.



(a) Guards are shown on the edges leading to the respective transitions. The guards and the underlying decision trees can be further explored by selecting the transition in the MPE. We manually added the labels of the invisible routing transition and we omitted the write operations and variable to improve the readability of the model.

Transition	Guard expression
Send Fine	totalPaymentAmount ≤ 18
Payment	totalPaymentAmount ≤ 18
Inv1	totalPaymentAmount > 18
Inv4	dismissal = G
Inv5	dismissal = NIL
Inv6	expense ≤ 15.6
Receive Result	expense > 15.6

(b) Tabular view of the discovered guards added for better legibility.

Figure 11.16: Process model with discovered guard expressions for the output transitions of the places $p10$, $p12$, and $p14$ by applying the overlapping decision mining method.

We switch the MPE to the **data discovery** mode (cf., configuration option ⑤ in Figure 11.2). In the data discovery mode, an additional configuration panel appears that allows users to configure the employed decision mining algorithm and its parameters. Figure 11.15 shows the configuration panel with the configuration that we use to discover the decision rules for place *pl12* of the road traffic fine management process. The configuration allows users to configure which guard-discovery algorithm to use, which data attributes to consider, and the configuration of the following parameters:

- the minimum number of instances at decision-tree leafs (*min instances*),
- the minimum fitness of trace to be considered (*min fitness*),
- whether to apply 10-times 10-fold cross validation, and
- whether to force the discovery of binary rules.

In our case, we choose our proposed overlapping decision mining approach (cf., Chapter 10) and set the *min instances* (*mi*) parameter to 0.01, i. e., the parameter is set of 1% of the instances reaching the decision point *pl12*. The *min instances* parameter influences whether the discovered guards are over-fitting (values too low), or under-fitting (value too high). The resulting process model that is enhanced with decision rules is depicted in Figure 11.16. We discovered decision rules for three of the places: *pl10*, *pl12*, and *pl14*. The actual decision rules are analyzed later in Chapter 12 as part of a case study. Here, we only use them to introduce the features of the MPE.

Step 3: Fitness and Precision Computation.

Having discovered or obtained a normative multi-perspective process model from other source, we can use the MPE to evaluate the quality of the discovered process model. This requires to change the MPE to its *fitness mode* and, subsequently, to its *precision mode*. Please observe that there is a substantial difference compared with the analysis done in the first step: now the model contains the data perspective, i. e., guard expressions are defined for several transitions. On average the newly discovered process model gets a fitness score of 0.996, i. e., the average fitness does not change considerably, and an average precision score of 0.695, i. e., the average precision was improved by 0.033 percentage points. The detailed diagnostics displayed by the fitness- are the precision mode are shown in Figure 11.17. In comparison to the fitness diagnostics for the input model (cf., Figure 11.13) the coloring of transitions did not change considerably. Only the first *Payment* and the *Send Fine* activity are diagnosed with a few more fitness problems. On a closer inspection, the local fitness score for both activities is almost 1.0. In comparison to the precision diagnostics for the input model (cf., Figure 11.14), the coloring of the places *pl10*, *pl12*, and *pl14* changed considerably. The guards discovered for the respective output transitions increased the local activity precision measure on these places. Please note that the activity precision for place *pl12* is not 1.0 since the discovered

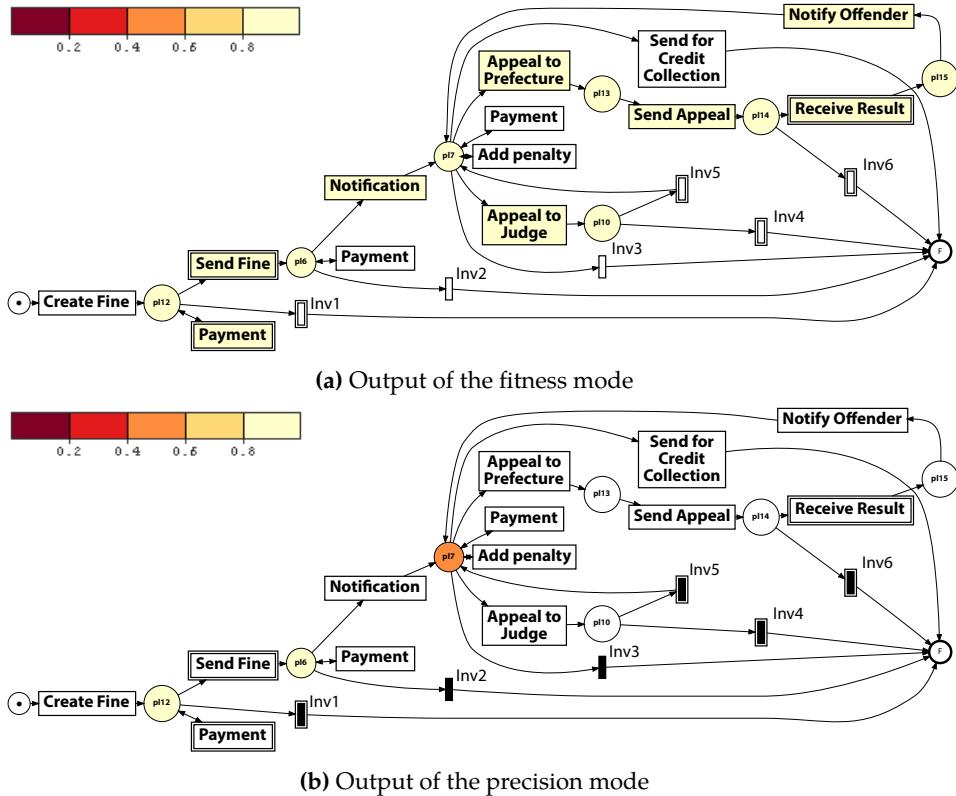


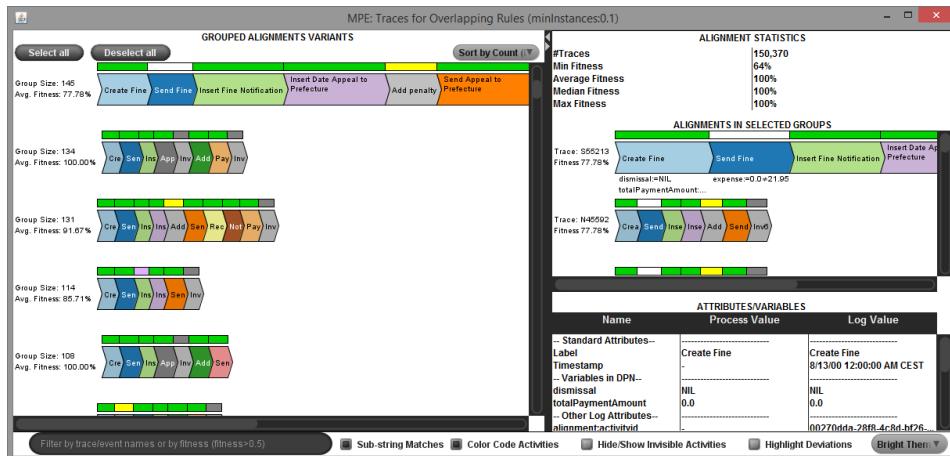
Figure 11.17: Quality diagnostics for the process model with discovered data perspective in terms of fitness and precision as shown by the MPE.

decision rules are overlapping.

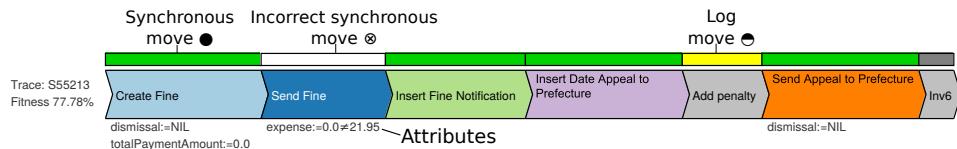
Step 3: Detailed Analysis Using the Trace- and Chart View.

Beyond analyzing fitness and precision of multi-perspective process models, the end user may want to explore specific traces and attributes of the event log in detail. For example, some traces may correspond to problematic process instances. Also, the distribution of some attributes may be potentially interesting; however, no decision rule could be discovered. To answer these questions, the MPE provides two complementary views on the process that allow to explore the event log using the process model in more detail: the **trace view** and the **chart view**.

Figure 11.18 shows the **trace view** that opens on a second screen upon pressing the *Toggle Trace View* button. For each log trace, the corresponding alignment is shown. Traces are grouped based on the sequence of activity executions (i.e., the same



(a) The trace view of the MPE. The left panel shows alignments grouped based on the same sequence of activities and the right panel shows individual alignments for the selected groups.



(b) Alignment of trace S55213 as displayed in the trace view of the MPE.

Figure 11.18: Trace view of the MPE showing details on the alignment of individual traces to the process model.

behavior in the control-flow perspective) on the left side of the window. Individual traces and their data attributes can also be explored on the right side of the window. Log and model moves are highlighted with yellow and purple color above the move, incorrect synchronous moves (i.e., a move with invalid data assignment) is highlighted with white color as shown for the move *Send Fine* in Figure 11.18b. Moves related to the same activity are painted with the same color. For example, all moves for the transition *Create Fine* are painted in light blue in Figure 11.18, which allow to quickly spot patterns across traces.

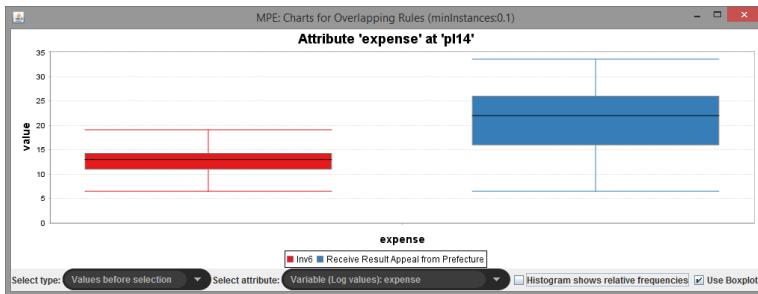


Figure 11.19: Chart view comparing the distribution of values of the attribute *expense* in the log projection of the alignment for the two output transitions of place *pl14*.

The **chart view**, shown in Figure 11.19, provides more details on the distribution of data attributes at certain states within the process model using three types of diagrams: histograms, box-plots, and bar-charts depending on the attribute type. Figure 11.19 shows a context-sensitive boxplot of the distribution of the values of attribute *expense* before the occurrence of the activities *Receive Result Appeal from Prefecture* (blue) and the invisible step *Inv6* (red), which are the two output transitions of place *pl14*. It is possible to show either the actual values observed in the log projection of the alignment or the values of the process projection of the alignment, which may be corrected to fulfill the guard expressions for incorrect synchronous moves. In Figure 11.19, the values observed in the log projection of the alignment are shown. This allows end users to visually analyze whether certain ranges of values are usually observed together with the occurrence of given activities. For instance, the distribution depicted illustrates the discovered decision rule for place *pl14*: cases with an expense of more than 15 EUR are more likely to continue the process with activity *Receive Result Appeal from Prefecture*.

11.2.3 Conclusion of the MPE

We presented the MPE as a tool for multi-perspective process exploration, which has reached a high degree of maturity. It has been used in several real-life case studies and on event logs of considerable size. This is also testified by its application

on the analysis of several real-life event logs that are presented in Chapters 12 to 15. For example, as part of the case study presented in Chapter 15, we applied it an event log with over 1,000,000 traces and more than 6,000,000 events. A limitation of the MPE is its reliance on the automatic graph layout provided by the ProM GraphViz package.⁴⁹ The automatic graph layout works well in simple cases, but produces poor layouts for cyclic structures and many variable nodes and guard expressions. Future work on better visualizations and better layout methods for complex DPNs is needed.

11.3 Conclusion

We presented the Multi-perspective Process Explorer (MPE) and the Interactive Data-aware Heuristic Miner (iDHM), two interactive tools that implement methods proposed in this thesis. Both tools are released as open-source software in the ProM framework and have been successfully used in several real-life settings as we show in the next four case-study chapters. Still there is a gap to available commercial tools with regard to the ease of use and the intuitiveness of the user interface. In particular both tools provide many parameters influencing the analysis result. A good direction for future work would be to provide better insights in the effect on the result that each of those parameters has. Finally, it would be beneficial to integrate the Guided Process Discovery (Chapter 9) method into the iDHM. Process analysts would be able to directly see how and if injected domain knowledge improves the discovered process.

⁴⁹The external software GraphViz (<http://www.graphviz.org>) is used to compute a good layout using the layout manager DOT.

12 Case Study: Road Traffic Fine Management

In this chapter and in the following three chapters, we present four case studies that we conducted to show that the proposed method are applicable in real-life situations, i. e., that it is feasible to apply them to real-life problems and that they provide valuable insights. In contrast to a classical case study, our emphasis here is on the validation of our methods. Therefore, we choose to structure the case studies along the proposed method instead of the business questions raised.

This chapter presents the *road traffic fine management fines* (abbreviated as *fine management*) case study. We conducted it in the context of the process of managing road traffic fines by a local police force in Italy. Parts of this case study were published in [Man+14].

We structure this chapter as follows. First, we describe the context of the case, specific process questions, the event log, and a hand-made normative process model in Section 12.1. Then, we describe how we applied each of the four proposed methods to the fine management case. We started by checking the conformance of the normative process model in Section 12.2. Afterwards, we enhanced the normative model with automatically discovered decision rules in Section 12.3. Then, we evaluated what results could be achieved using a purely data-driven approach, i. e. without the normative model. We discovered and analyzed process models using both our DHM method (Section 12.4) and our GPD method (Section 12.5).

12.1 Case Description

We analyzed the process of managing road traffic fines by a local police force in Italy. An ad-hoc information system that is purpose-built supports the management and handling of their road traffic fine management process. This information system is not process-aware, i. e., there is no formal specification on how it should be used to manage the road traffic fines. Also, the documentation does not provide a description of the process. However, we were able to interview stakeholders to obtain some information on the process. In the next three subsection, we are describing the process questions that were formulated (Section 12.1.1), elaborate on the extracted event log and the semantics of the events (Section 12.1.2), and present a normative process model that we designed based on our knowledge of the process (Section 12.1.3).

Table 12.1: Activities recorded in the fine management event log.

Activity	Frequency	Description
Create Fine	150,370	The initial creation of the fine in the information system.
Send Fine	103,087	A notification about the fine is sent by post to the offender.
Insert Fine Notification (Notification)	79,860	The notification is received by the offender.
Add Penalty	79,860	An additional penalty is applied.
Payment	77,601	A payment made by the offender is registered.
Send For Credit Collection	59,013	A separate credit collection process is started for unpaid fines.
Insert Date Appeal to Prefecture (Appeal to Prefecture)	4,188	The offender appeals against the fine to the prefecture.
Send Appeal to Prefecture (Send Appeal)	4,141	The appeal is sent to the prefecture by the local police.
Receive Result Appeal from Prefecture (Receive Result)	999	The local police receives the result of the appeal.
Notify Result Appeal to Offender (Notify Result)	896	The local police informs the offender of the appeal result.
Appeal to Judge	555	The offender appeals against the fine to a judge.

12.1.1 Process Questions

Even though there system and the documentation does not provide a description of the underlying process, the managing of road traffic fines is regulated by the Italian law. Thus, there are some constraints that the process needs to adhere to. The main question of the case study was whether the real execution of the process as recorded by the information system matched these constraints (e.g., the fine is sent out in due time). Moreover, we also checked whether the assumptions of the stakeholders on the process as represented in our normative model (e.g., the entire fine amount is paid) were correct. We present the respective assumptions and constraints later in Section 12.1.3 when describing the normative model.

12.1.2 Event Log

The information system records data about its operations in a PostgreSQL database. We obtained a snapshot of the database that was taken in June 2013 and converted it to a XES standard compliant event log [IEEECIS16]. The resulting event log was made available for further process mining research purposes as part of the collection of real-life event logs of the IEEE Task Force on Process Mining [LM15].⁵⁰

The event log contains 150,370 traces and 561,470 events that were recorded between January 2000 and June 2013. Most of the traces are short: on average, a

⁵⁰The road fines event log can be obtained from <https://doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5>

Table 12.2: Attributes recorded in the fine management event log.

Attribute	Shorthand	Domain	Description
amount	am	continuous	The amount due to be paid for the fine.
article	ar	discrete	The number of the article of the Italian road-traffic law that is violated by the offender.
dismissal	di	literal	A flag indicating whether and how the fine is dismissed. Several values are possible: G encodes a dismissal by the judge, # encodes that the fine was dismissed by the prefecture, and NIL encodes that the fine was not dismissed. There are several other values used for which we cannot reveal the semantics.
expense	ex	continuous	The additional amount due to be paid for postal expenses.
notificationType	nt	literal	A code indicating to whom the fine refers. The codes used in the event log are P (the owner of the car) or C (the driver that committed the offense). When the actual offender is unknown (e.g., was not stopped), a fine is created for the owner of the car.
org:resource	or	literal	A code indicating the employee who handled the case. We cannot disclose more information regarding these codes.
paymentAmount points	py po	continuous discrete	The amount paid by the offender in one transaction. The penalty points deducted from the offender's license. In Italy each driver starts with 20 points on their license and may lose points for each offence. Drivers who lose all their points need to take a new driving test.
totalPaymentAmount (payment)	pa	continuous	The cumulative amount paid by the offender.
vehicleClass	vc	literal	The kind of vehicle used by the offender.

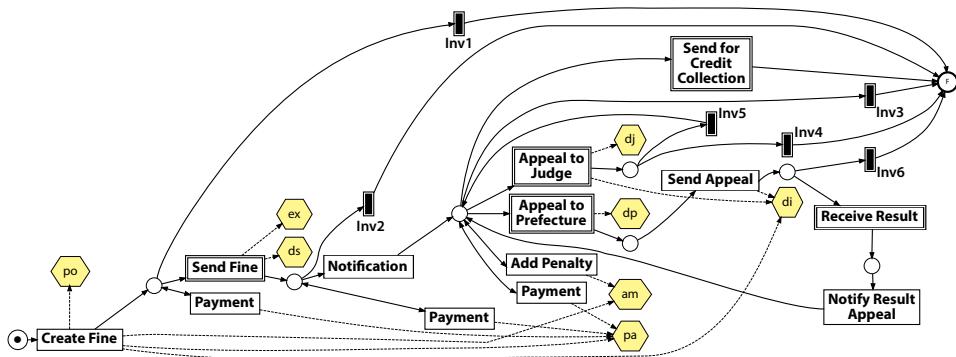
trace consists of four events only. The longest trace consists of 20 events. The event log contains 11 different activities that are conducted as part of managing the road traffic fines. Table 12.1 lists the activities in the order of their frequency and provides a brief description of each activity. Moreover, 12 data attributes are recorded. The attributes relate to the fine amount due, the payments made by the offender, and, possibly, the outcome of appeals against the fine. Table 12.2 shows the attributes. For readability, we shorten the names of some activities and variables as shown in Tables 12.1 and 12.2.

From a basic analysis of the event log, we noticed that cases are usually completed within 6 months, including those cases that end with a referral to credit collection. For the analysis, we wanted to consider only finished cases. As a heuristic to ensure that we only retained finished cases, i.e., we filtered out any case that started after June 2012. Since the relevant laws and procedures are rather stable over the past years, we assumed that the last year of the event log shows the same behavior as in previous years. The resulting filtered event log contains 145,800 log traces and 543,538 events, which were recorded between January 2000 and June 2012. For 43% of the traces the process ends after two events: the fine is either directly paid (*Payment*) or the letter with information about the fine is sent out (*Send Fine*) but

the fine is not yet paid. In contrast to this simple part of the log, 51% of the traces recorded five or more events and 62% of the cases take longer than 100 days to finish. This suggests that many offenders did not pay the fine in time or appealed against the decision.

12.1.3 Normative Process Model

We manually created a normative DPN that encodes the Italian laws regarding the management of road fines. The DPN is shown in Figure 12.1a and its guards are listed in Figure 12.1b.



(a) DPN of the road fines management process. We use the abbreviated from of variables.

Transition	Guard
Send Fine	delaySend' < 90 days
Appeal to Judge	delayJudge' < 60 days
Appeal to Prefecture	delayPrefecture' < 60 days
Receive Result	dismissal = NIL
Send for Credit Collection	payment < amount + expenses (dismissal ≠ NIL) ∨ (payment ≥ amount ∧ points = 0)
Inv1	
Inv2	payment ≥ amount + expenses
Inv3	payment ≥ amount + expenses
Inv4	dismissal = G
Inv5	dismissal = NIL
Inv6	dismissal = #

(b) Guards expression function of the DPN

Figure 12.1: The normative DPN created for the road fines management process.

The process starts with the *Create Fine* transition that writes four variables **amount**, **points**, **payment**, and **dismissal**. Variable *amount* refers to the amount that needs to be paid by the offender. Variable *points* records the number of points that are deducted from the offender's license. Variable *payment* specifies the total amount that has been paid by the offender, i. e., the variable corresponds to the log attributes

totalPaymentAmount. It is always initialized as *payment* = 0.0. Variable *dismissal* stores a character that encodes the reasons why a certain fine was dismissed. A value of NIL encodes that the fine was not dismissed (i. e., it has to be paid). In general, the offender can pay the fine (partly or fully) at many moments in time: Right after the creation, after a road fine notification is sent by the police to the offender's place of residence, or when such a notification is received by the offender. If the entire amount is paid (or, even, by mistake, more than that amount), the fine management is closed. This motivates the presence of the invisible transitions *Inv1*, *Inv2*, and *Inv3*. If a notification is sent, the offender needs to also pay the postal expenses.

If the offender does not pay within 180 days, a penalty is added, which is usually as much as the initial fine's amount. After being notified by post, the offender can appeal against the fine through a judge and/or the prefecture. If the appeal is successful, the variable *dismissal* is set to value G or #, respectively, and the case ends (by firing either *Inv4* or *Inv6*).⁵¹ Otherwise, the case continues by firing *Inv5* or *Receive Result Appeal*. If the offender does not pay, eventually the process ends by handing over the case for credit collection.

The Italian laws specifies a number of time constraints. The fine notification must be sent within 90 days since its creation. After the notification, the offender may only appeal to a judge/prefecture within 60 days. To check the conformance of the fine management with respect to these laws, we have preprocessed the event log and introduced three additional variables that record the various delays: *delaySend*, *delayJudge*, and *delayPrefecture* with the respective shorthand notations *ds*, *dj*, and *dp*.

12.2 Conformance Checking

In this section, we describe how we applied the balanced multi-perspective conformance checking method (cf., Chapter 5) to the normative process model and the fine management event log.

12.2.1 Configuration Settings and Cost Function

Computing an optimal alignment for conformance checking requires the definition of a cost function κ (cf., Definition 4.3). Since the fine's amount and the deducted points are defined by law and the expenses follow the Italian post tariffs, their values cannot be modified to give an explanation of deviations. In order to respect this domain characteristic, we assigned significantly higher costs to their deviations in comparison to those for deviations of the values of *Payment* and *Dismissal* as well

⁵¹In [Man+16a], we mistakenly swapped the meaning of values G and # for the fine dismissal in the normative process model.

Table 12.3: Cost function κ for the fine management process. The cost for a *synchronous move* is 0 for all transitions. Log moves for invisible transitions are assigned an infinite cost. The cost for an incorrect synchronous move is obtained by summing the specific costs associated with each variable for which a missing or incorrect value has been written.

Transition	Model Move	Log Move	Variable	Missing	Incorrect
Create Fine	1	1	delaySend (ds)	1	1
Send Fine	1	1	delayPrefecture (dp)	1	1
Payment	1	1	delayJudge (dj)	1	1
Notification	1	1	amount (am)	1	3
Add Penalty	1	1	expense (ex)	1	3
Appeal to Judge	1	1	payment (pa)	1	3
Appeal to Prefecture	1	1	points (po)	1	3
Send for Credit Collection	1	1	dismissal (di)	1	1
Send Appeal	1	1			
Receive Result	1	1			
Notify Offender	1	1			
Inv1, ..., Inv6	0	∞			

as the control flow. This ensures that these values are never varied while building an alignment.

The employed cost function is specified by Table 12.3 according to Definition 4.3. A tabular representation is used to enhance the readability. A cost for a model move $((\gg, (t, w))$ for a transition t is equal to the cost for t in the left-hand side table plus the cots of the missing writing of variables in w in the right-hand side table. Similarly, the cost for a log move (e, \gg) for an event e equal to the cost of $\#_{act}(e)$ in the left-hand side table. Finally, the cost for an incorrect synchronous move for $((e, (t, w))$ for an event e and a transition t is equal to the sum of the cost of incorrect write operations as shown in the right-hand side table.

As an example, consider an alignment move $(e, (t, w))$ in which event e recorded the execution of activity *Create Fine*, i.e., $\#_{activity}(e) = \lambda(t) = CreateFine$ and two out of the four prescribed write operations recorded incorrect values, e.g., for both variable v_1 with $v(v_1) = di$ and variable v_2 with $v(v_2) = am$ the value is incorrect: $w(v_1) \neq \#_{v(v_1)}(e)$, $w(v_2) \neq \#_{v(v_2)}(e)$. Then, the cost according to the cost function specified in Table 12.3 is: $\kappa((e, (t, w))) = 1 + 3$.

12.2.2 Conformance Checking Results

We applied both the balanced alignment method presented in Chapter 5 as well as the non-balanced alignment method that is described in [LA13a]. Both methods were applied using the same cost function, which was just described, as parameter. We show that it is necessary to use the balanced alignment method that guarantees an optimal alignment with regard to a cost function defined for all perspectives of the process.

First, we present the result of the balanced alignment method. Figure 12.2 shows

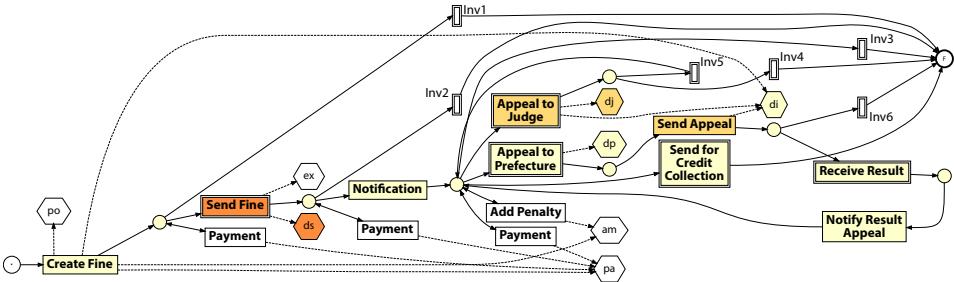


Figure 12.2: Conformance diagnostics projected on the road fine traffic management DPN as produced by the MPE fitness mode for the balanced alignment method.

the resulting projection of conformance problem onto the model in the fitness mode of the MPE (cf., Section 11.2 for the calculation of local fitness measures used to color the places and transitions). The overall fitness score was 0.96, which testifies a very good conformance of the event log with the process model. In particular, 53.1% of the log traces are characterized by a fitness level of 1.0.

However, as the fitness projection indicates, several deviations are still present. For example, transitions *Send Fine* and *Appeal to Judge* are colored with the darkest color. The deviations are mainly due to the constraints on the time perspective of the process. In 25.9% of all traces, the value written to *delaySend* by transition *Send Fine* is incorrect, i. e., the notification is sent to late. Moreover, in 20.4% of the traces in which an appeal to a judge is made (*Appeal to Judge*), the value of *delayJudge* is incorrect. This suggests that authorities are currently unable to handle road fines in a timely and correct manner. Therefore, to remedy this situation, more resources, i. e., police officers, should be assigned by the municipality. Alternatively, some parts of the management should be outsourced, e. g., the steps necessary to print fine notifications, put these in envelops, and send them by post. Indeed, these are manual steps that require a lot of time from the involved police officers.

A valuable insight is that there are deviations recorded for the *Send for Credit Collection* transition. In 8.2% of all traces the transition appears as a *model move* in the alignment. For 8.2% of the fines, within 1 year, neither have their amount been paid in full nor have they been forwarded for credit collection. Considering that sending for credit collection is supposed to usually occur within 6 months after the fine has been opened, this finding suggests that there may be issues (e. g., unmotivated delays) with managing unpaid fines.

12.2.3 Comparison With the Non-Balanced Method

In order to compare the results of our balanced method with those returned by a non-balanced method [LA13a], we also applied the latter. For the comparison

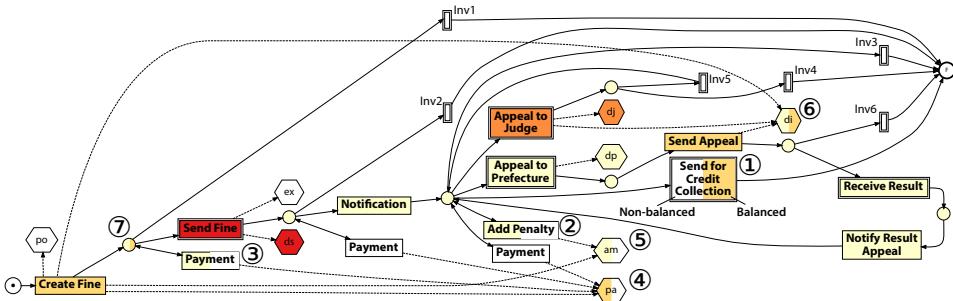


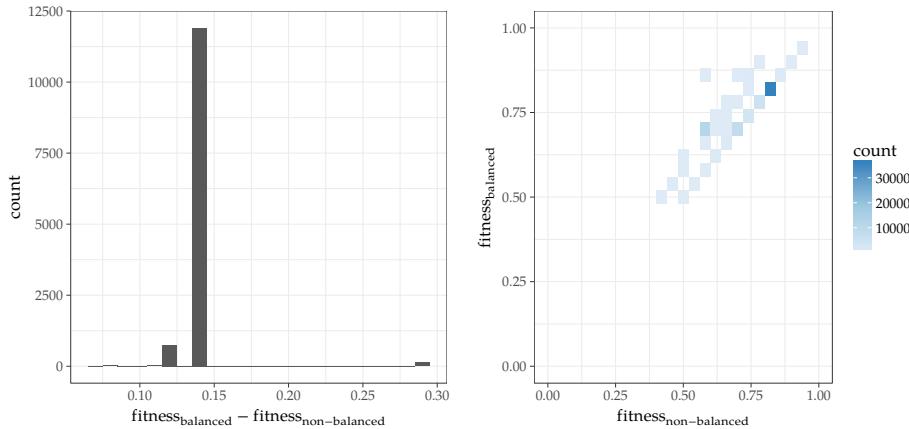
Figure 12.3: Comparison of the *MPE fitness mode* output for both methods. The color-coding returned for the non-balanced method is shown on the left side of transitions and variables and the color-coding returned for the balanced method is shown on the right side. The darker the color, the higher the percentage of detected deviations for the transition or the variable. The figure clearly shows that the balanced method provides different results suggesting that the method to first construct control-flow alignments may provide misleading results.

we use only the non-perfectly fitting traces (68,330), since there is no difference in fitness to be expected for perfectly fitting traces⁵².

First, we directly compared the difference in fitness level between both methods in Figure 12.4. Section 12.2.3 shows that for a large group of about 12,000 traces, the difference in fitness between the balanced method and the non-balanced method is more than 0.14. For a smaller group of 117 traces the difference in fitness level is even 0.29. Section 12.2.3 indicates that the balanced method improved the fitness for many traces in the whole range of fitness values. The fitness improved for every trace that is placed in a box above the diagonal.

Second, we compare the output of the *MPE fitness mode* for both the balanced and the non-balanced method in Figure 12.3 to give an indication on where the differences are. The color-coding returned by the non-balanced method is shown on the left side of transitions, places, and variables and the color-coding returned by the balanced method is shown on the right side. The comparison shows that there are seven larger differences (1-7) in the identification of the root-causes of the deviations. In particular, when applying the non-balanced method, the net projection highlights that many traces are deviating due to wrong values of variables **amount** (5) and **payment** (4). Indeed, the color on the left side of these variables is yellow and dark yellow. Vice versa, the right side of the corresponding variables is white-colored. Regarding the variable *amount*, according to the Italian law any root cause that consists of changing the assignment of such a data variable (i. e.,

⁵²Generally, the non-balanced method may align even a perfectly fitting control-flow such that it results in a data-based deadlock, such as described in Section 5.3. However, this did not occur in the road fines case.



(a) Histogram of the difference in fitness level between both methods. (b) Binned scatterplot of the fitness returned by both methods.

Figure 12.4: Comparison between the fitness level computed by the balanced and non-balanced method.

an incorrect write operation) is not acceptable. After all, this amount is defined by the Italian law and police officers use road-fine forms in which the amount is predetermined.

Table 12.4: Exemplary trace σ_A with an unpaid fine.

id	activity	am	ex	po	pa	di	ds	dj
e ₁₀	Create Fine	131.0		0	0	NIL		
e ₁₁	Send Fine		10.0				1152	

Third, in order to understand the reason for the difference in the identification of root causes, we have inspected the individual alignments returned by the two methods. We found out that there are alignments for hundreds of log traces of the form of σ_A , as shown in Table 12.4. Figure 12.5 compares the output of the trace view diagnostics of the Multi-perspective Explorer (MPE) for the trace σ_A when using the balanced and non-balanced method. Moreover, for completeness, we show both alignment in a tabular form in Table 12.5. The non-balanced method highlights that the fine at creation time should have already been associated with a payment of 5,000 EUR. This is in contrast with the balanced method which suggests that the fine should have been dismissed with any code different from NIL (e.g., here our alignment method arbitrarily choose the code ANY) and never been sent out. Thus, the non-balanced method diagnoses more deviations regarding the



(a) The non-balanced alignment (fitness 0.77): *Create Fine* is marked as incorrect synchronous move (white) and *Send Fine* as correct synchronous move (green).

(b) The balanced alignment (fitness 0.85): *Create Fine* is marked as incorrect synchronous move (white) and *Send Fine* as log move (yellow).

Figure 12.5: Output of the trace view diagnostics of the MPE for trace σ_A

Table 12.5: Non-balanced and balanced alignment for the log trace σ_A .

(a) Non-balanced alignment with a fitness-score of 0.77.

Event attributes (#(e))	Move ($e, (t, w)$)	Process variables (w)
(am \mapsto 131.0, pa \mapsto 0.0, po \mapsto 0, di \mapsto NIL) (ex \mapsto 10.0, ds \mapsto 1152)	$\otimes (e_{10}, (\text{Create Fine}, w'_{10}))$ $\bullet (e_{11}, (\text{Send Fine}, w'_{11}))$ $\bullet (\gg, (\text{Inv}2, \emptyset))$	(am \mapsto 131.0, pa \mapsto 5000.0, po \mapsto 0, di \mapsto NII) (ex \mapsto 10.0, ds \mapsto 1152)

(b) Balanced alignment with a fitness-score of 0.85.

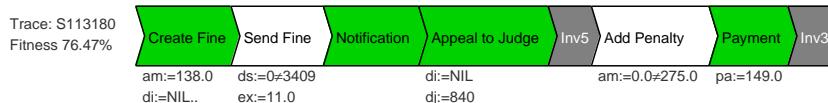
Event attributes (#(e))	Move ($e, (t, w)$)	Process variables (w)
(am \mapsto 131.0, pa \mapsto 0.0, po \mapsto 0, di \mapsto NIL) (ex \mapsto 10.0, ds \mapsto 1152)	$\otimes (e_{10}, (\text{Create Fine}, w_{10}))$ $\bullet (e_{11}, (\text{Send Fine}, \emptyset))$ $\bullet (\gg, (\text{Inv}1, \emptyset))$	(am \mapsto 131.0, pa \mapsto 0.0, po \mapsto 0, di \mapsto ANY)

variable payment (④) than the balanced method whereas the balanced method indicates more deviations regarding the variable *dismissal* (⑥) and log moves for *Send Fine* (⑦) than the non-balanced method. This case, the alignment returned by the non-balanced method is not plausible from a domain perspective, since it should be impossible to create a fine that already has a full payment associated to it. According to our domain knowledge of the process, it is more likely that the dismissal code was not correctly recorded at the start of the process. The payment by necessity can only be made at a later stage. We encoded this domain knowledge in the employed cost function, which was accounted for by the balanced method, but not by the non-balanced method.

Moreover, there is a smaller number of traces like trace σ_B , as shown in Table 12.4. Trace σ_B also has conformance problems. First, the fine was sent too late, since the delay is longer than what law permits. Second, the fine has been closed with a payment of 149 EUR, which corresponds to the initial amount 138 EUR plus the postal expenses of 11 EUR. Unfortunately, a penalty was added to the fine, which the offender did not pay. Therefore, the fine management should not have been closed.

Table 12.6: Exemplary trace σ_B with an underpaid fine.

id	activity	am	ex	po	pa	di	ds	dj
e ₂₀	Create Fine			138.0		6	0	NIL
e ₂₁	Send Fine			11.0				3409
e ₂₂	Notification							
e ₂₃	Appeal to Judge						NIL	
e ₂₄	Add Penalty							840
e ₂₅	Payment	275.0				149.0		



- (a) The non-balanced alignment with a fitness score of 0.76 marks *Send Fine* and *Add Penalty* as incorrect synchronous move (white).



- (b) The balanced alignment with a fitness score of 0.88 marks *Send Fine* as incorrect synchronous move (white) and *Send for Credit Collection* as model move (purple).

Figure 12.6: Output of the trace view diagnostics of the MPE for trace σ_B

As shown by the trace view diagnostic output of the MPE Figure 12.6 and the tabular listing of the alignments in Table 12.7, both methods highlight the problem that the fine is sent too late. The observed value for variables *delaySend* (ds) is 3,409 hours but according to the process model, the fine should have been sent out within 90 days, i. e., 2,160 hours. There is a second observation that does not conform to the behavior prescribed by the process model: the payment of 149 that was made does not cover the full fine amount. For this second source of non-conformance, the non-balanced method suggests that, after applying the penalty, the due amount is set to zero. This results in a higher number of deviations regarding variable *amount* reported by the non-balanced method compared to the balanced method (cf., ⑤ in Figure 12.3). This is definitely not plausible since adding a penalty needs to result in a higher amount to be paid. As a matter of fact, the Italian law states that, besides very few exceptions, the due amount should even be doubled, excluding expenses. By contrast, the balanced method returns a meaningful result: The fine was not paid in full and, hence, needs to be sent for credit collection. Therefore, the balanced method reports more deviations for the transition *Send for Credit Collection* (cf., ⑥ in Figure 12.3) compared to the non-balanced method.

The reason for the differences in the returned alignments is related to the fact that the non-balanced method constructs alignments by initially aligning the control-flow and, only later, by aligning the other perspectives. The non-balanced method makes the assumption that control-flow deviations are more costly and, hence, they can be aligned first. If this assumption does not hold, such as for this case study, the returned alignments are not optimal and this may lead to implausible explanations.

We conclude this section by briefly reporting on the execution time of both methods. Finding the alignments took on average 4.8 milliseconds per trace for the balanced method, versus 1.3 milliseconds for the non-balanced one. The balanced method required more time but is justified by the more meaningful explanations for the deviations.

Table 12.7: Non-balanced and balanced alignment for the log trace σ_B .

(a) Non-balanced alignment with a fitness-score of 0.76.

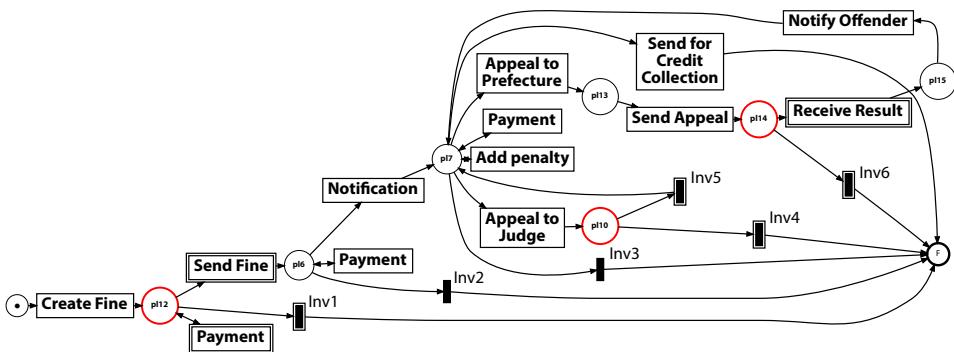
Event attributes (#(e))	Move (e, (t, w))	Process variables (w)
(am \rightarrow 138.0, pa \rightarrow 0.0, po \rightarrow 0, di \rightarrow NIL)	\otimes (e ₂₀ , (Create Fine, w' ₂₀))	(am \rightarrow 138.0, pa \rightarrow 0.0, po \rightarrow 0, di \rightarrow NIL)
(ex \rightarrow 11.0, ds \rightarrow 3409)	● (e ₂₁ , (Send Fine, w' ₂₁))	(ex \rightarrow 11.0, ds \rightarrow 0)
\emptyset	● (e ₂₂ , (Notification, \emptyset))	
(di \rightarrow NIL, dj \rightarrow 840)	● (e ₂₃ , (Appeal to Judge, w' ₂₃)) ◐ (>, (Inv5, \emptyset))	(di \rightarrow NIL, dj \rightarrow 840)
(am \rightarrow 275.0)	\otimes (e ₂₄ , (Add Penalty, w' ₂₄))	(am \rightarrow 0.0)
(pa \rightarrow 149.0)	● (e ₂₅ , (Payment, w' ₂₅)) ◐ (>, (Inv3, \emptyset))	(pa \rightarrow 149.0)

(b) Balanced alignment with a fitness-score of 0.88.

Event attributes (#(e))	Move (e, (t, w))	Process variables (w)
(am \rightarrow 138.0, pa \rightarrow 0.0, po \rightarrow 0, di \rightarrow NIL)	\otimes (e ₂₀ , (Create Fine, w ₂₀))	(am \rightarrow 138.0, pa \rightarrow 0.0, po \rightarrow 0, di \rightarrow NIL)
(ex \rightarrow 11.0, ds \rightarrow 3409)	● (e ₂₁ , (Send Fine, w ₂₁))	(ex \rightarrow 11.0, ds \rightarrow 0)
\emptyset	● (e ₂₂ , (Notification, \emptyset))	
(di \rightarrow NIL, dj \rightarrow 840)	● (e ₂₃ , (Appeal to Judge, w ₂₃)) ◐ (>, (Inv5, \emptyset))	(di \rightarrow NIL, dj \rightarrow 840)
(am \rightarrow 275.0)	● (e ₂₄ , (Add Penalty, w ₂₄))	(am \rightarrow 275.0)
(pa \rightarrow 149.0)	● (e ₂₅ , (Payment, w ₂₅)) ◐ (>, (Send for Credit Collection, \emptyset))	(pa \rightarrow 149.0)

12.3 Discovery of the Data Perspective

We also checked whether additional rules can be discovered that are not described in the normative model. Therefore, we applied our overlapping decision mining method (cf., Chapter 10) on the normative process model. We used a Petri net version of the DPN shown in Figure 12.1a, i. e., all existing guards and variable write operations were removed. We employed the MPE *data discovery* mode with the same configuration settings as described in Section 11.2 and evaluated the quality of the resulting multi-perspective model with both our precision measure (cf., Chapter 6) and our fitness measure (cf., Section 4.4).



(a) Decision rules have been discovered for the places of the DPN that are highlighted in red. The guards listed in Figure 12.7b have been assigned to transitions with a double-border.

Transition	Guard expression
Send Fine	totalPaymentAmount ≤ 18
Payment	totalPaymentAmount ≤ 18
Inv1	totalPaymentAmount > 18
Inv4	dismissal = G
Inv5	dismissal = NIL
Inv6	expense ≤ 15.6
Receive Result	expense > 15.6

(b) Tabular view of the discovered guard expression.

Figure 12.7: Process model enhanced using the overlapping decision mining method.

The process model enhanced with decision rules is depicted in Figure 12.7.⁵³ In total, we discovered decision rules for three of the places: $p10$, $p12$, and $p14$. The decision rule discovered for transitions $Inv4$ and $Inv5$ (place $p10$) is mutually exclusive and confirms the rule specified in the normative process model (cf., Figure 12.1). The decision rule regarding transitions $Receive\ Result$ and $Inv6$ is also

⁵³Figure 12.7 shows the same process models as in Figure 11.16, which was used for the introduction of the data discovery feature of the MPE.

mutually-exclusive and suggests that appeals to the prefecture with low postal expenses are more often successfully appealed against. Finally, the guards discovered for the transitions *Send Fine*, *Payment*, and *Inv1* are overlapping. Note that this rule cannot be returned by state-of-the-art decision mining techniques [LA13b]. From a domain point-of-view the overlap of the different guards is plausible since both the *Payment* activity and the *Send Fine* activity are likely to be executed for those fines which have not yet been paid in full (i.e., less than 18 EUR has been paid). Conversely, the routing transition *Inv1* should only be executed for those fines that have been paid.

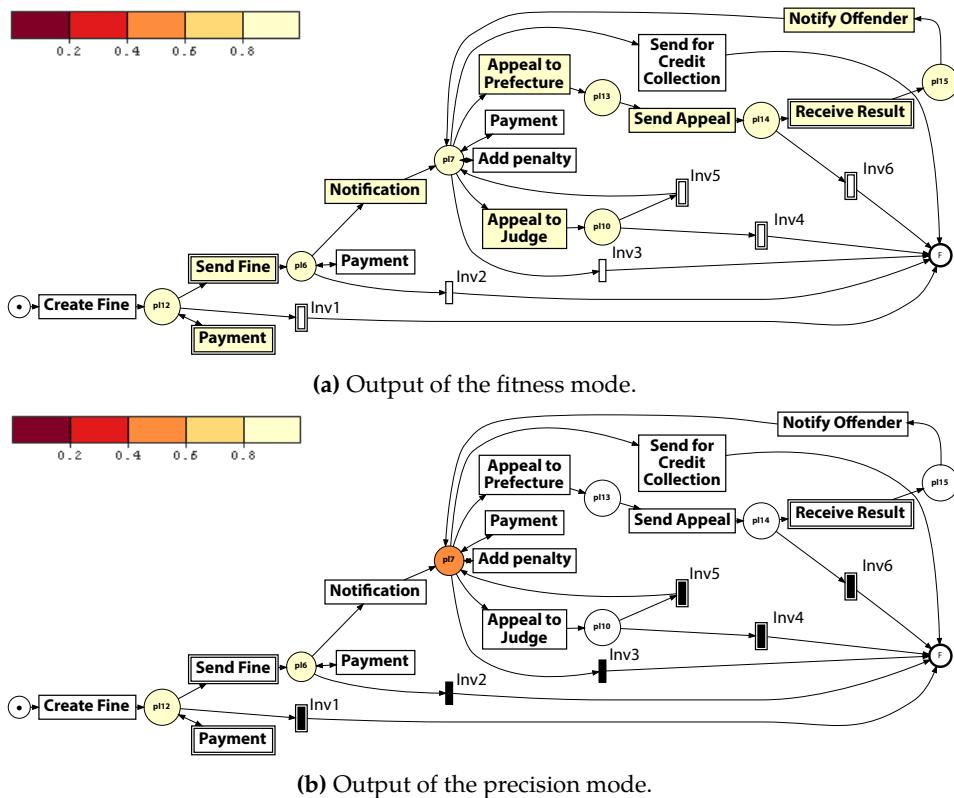


Figure 12.8: Quality diagnostics for the enhanced model as shown by the MPE.

We computed the multi-perspective precision and fitness scores for the process model enhanced with decision rules. On average the fitness of the process model with decision rules was 0.996, i.e., fitness did not change considerably, and the precision score is 0.695, i.e., precision improved by 0.033. The detailed diagnostics displayed by the fitness and the precision mode of the MPE are shown in Figure 12.8.

In comparison to the fitness diagnostics for the input model (cf., Figure 12.8a) the coloring of transitions did not change considerably. The local fitness only decreased for the first *Payment* transition at place $pl12$ and the *Send Fine* activity. However, on a closer inspection the score is still almost 1.0 for both activities. In comparison to the precision diagnostics for the input model (cf., Figure 12.8b), the coloring of the places $pl10$, $pl12$, and $pl14$ changed considerably. The guards discovered for the respective output transitions increased the local activity precision measure on these places.

12.4 Data-aware Process Discovery

Next, we investigated the results that could be achieved without using the normative process model. We applied the data-aware process discovery method (cf., Chapter 8) to the fine management event log and discovered a DC-Net that reveals infrequent data-dependent paths.

12.4.1 Configuration Settings

We used the following parameter settings for the iDHM (cf., Section 11.1) that we experimentally determined:

- the *all-task-connected heuristic*, since we know that each activity is of interest;
- the observation threshold $\theta_{obs} = 0.01$ (i. e., relations that appear in more than 1% of the log trace), because we want to capture the main flow of the process;
- the dependency threshold $\theta_{dep} = 0.8$ to discover only strong dependencies;
- the binding threshold of $\theta_{bin} = 0.001$ to exclude very infrequent bindings.

Moreover, we used eight of the attributes to discover dependency conditions and guarded bindings.⁵⁴ We used C4.5 with 10 times 10-fold cross validation and only accepted classifications with a conditional dependency measure of $\theta_{con} \geq 0.5$ (cf., Section 8.3.1). Thus, only good classifiers that generalize well are used to add conditional dependencies. We also used C4.5 to discover the binding guards. Here, we used F1-score as an evaluation measure, since the bindings are not necessarily infrequent. We included those guards that obtain an F1-score of at least 0.8.

12.4.2 Discovery Results

Figure 12.9 shows the DC-Net discovered by the iDHM for the fine management event log. The model was discovered within four seconds of computation time.

⁵⁴Specifically, we included the following attributes: amount, article, dismissal, expense, isPaid, paymentAmount, points, and totalPaymentAmount. We added the boolean attribute *isPaid* that encodes whether the fine is paid in full, i. e., *isPaid* is true if $totalPaymentAmount \geq amount$ and false, otherwise. This attribute is of interest for the analysis of the process since a full payment is desirable.

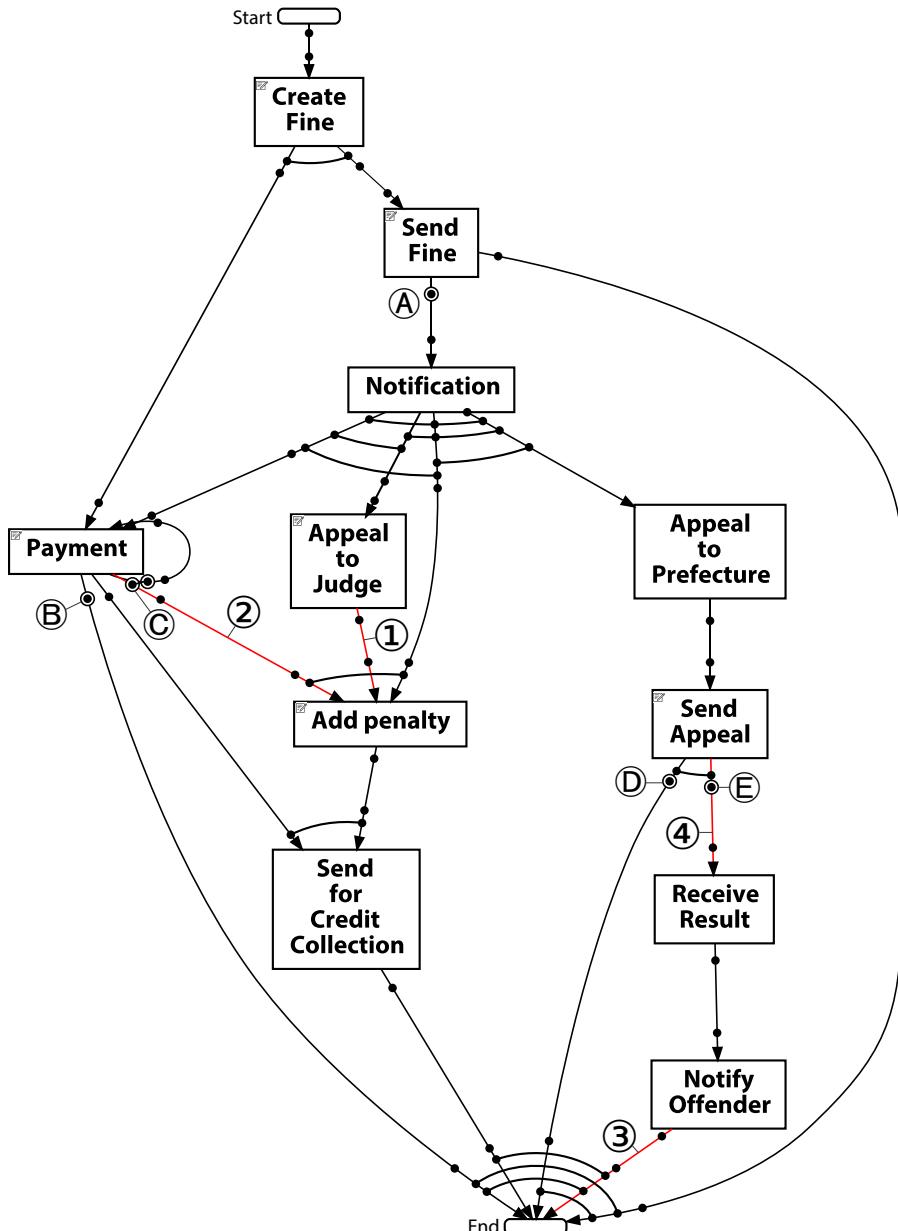


Figure 12.9: DC-Net discovered for the fine management event log. The numbered red edges (①–④) were added by our DHM method and the double-bordered bindings (Ⓐ–Ⓔ) are assigned the guards listed in Table 12.9.

Conditional Dependencies

Our method reveals three additional dependencies (red edges) compared to model returned by the standard Heuristics Miner. We numbered the additional dependencies in Figure 12.9 as ①–④. Table 12.8 lists the conditional dependency measure, the frequency of occurrence according to the binding heuristic, as well as quality-score (i.e., Cohen’s kappa) and used attributes of the obtained dependency condition for each relation. The computation time to discover the DC-Net with the iDHM was 12 seconds.

Table 12.8: Dependency conditions discovered for the fine management event log. We list the frequency with which the binding was activated according to the binding heuristic, the quality score of the dependency condition (i.e., Cohen’s kappa), and the dependency measure of the relation.

Nr	Source	Target	Frequency	Quality	Dependency	Attributes
①	Appeal to Judge	Add Penalty	275	0.85	0.93	amount, article, dismissal, points
②	Payment	Add Penalty	3,363	0.89	1	amount, isPaid
③	Notify Result	End	62	0.52	0.98	dismissal, expense
④	Send Appeal	Receive Result	509	0.82	0.97	dismissal, expense

Added dependencies ① and ②. The first two dependencies target both target activity *Add Penalty* and have a quality score (i.e., Cohen’s kappa) above 0.8, which can be considered as very good. The decision rule for dependency ① from *Appeal to Judge* to *Add Penalty* mainly depends on the value of the *dismissal* attribute. Cases with values G do not receive a penalty, whereas cases with value NIL receive a penalty. Next to the value of the *dismissal* attributes, this rule also depends on the fine amount, the number of points, and the article. According to the normative process model this is to be expected as those cases are dismissed by the judge. Dependency ② from *Payment* to *Add Penalty* is mainly based on the attribute *isPaid*. Unpaid fines with a small *amount* of less than 35 EUR receive a penalty.

Added dependency ③. Dependency ③ from *Notify Result* to the artificial activity *End* was discovered for cases with a *dismissal* value of # or G. The quality score of the classifier is 0.52 and, thus, slightly worse than the quality score of dependencies ① and ②. However, a value of above 0.5 for Cohen’s kappa can still be considered as acceptable for our purpose [LK77]. It is to be expected that the process finishes for cases with this code, since these cases are dismissed by the prefecture. Interestingly, this relation also occurs for cases with a *dismissal* value of NIL and high postal expenses. This should not happen, since these fines still need to be paid according to the normative process model.

Added dependency ④. Dependency ④ from *Send Appeal* to *Receive Result* was discovered based on a dependency condition that is based on the attributes *expense* and *dismissal*. The quality score of the dependency condition is high. The obtained classification rule indicates that this dependency can be predicted for cases with postal expenses above 19.85 Euros and the dismissal value of NIL, which encodes that the appeal was not successful. By further inspecting the dependency condition and the event log, we discovered that in a large number of cases the processing of the fine ends with the activity *Send Appeal to Prefecture* if the recorded dismissal value was different from NIL. Thus, differently than prescribed in the normative process model, the activity *Receive Result* is not logged for these cases.

In summary, the iDHM revealed four infrequent data dependencies that give more insights into the recorded behavior without obstructing the process model with infrequent dependencies that cannot be explained through data attributes.

Guarded Bindings

In addition to the conditional dependencies, we also discovered several guarded bindings by applying the decision mining and including discovered guards that exceed the configured decision quality threshold. Table 12.9 lists the discovered guard expressions that are highlighted using the letters ①–⑥ in Figure 12.9.

Table 12.9: Decision rules discovered for the guarded bindings.

Nr	Source	Target	Guard Expression
①	Send Fine	Notification	$\text{totalPaymentAmount} \leq 30.0$
②	Payment	End	$\text{isPaid} = \text{true}$ $\vee (\text{paymentAmount} \leq 18.78 \wedge \text{amount} > 35.0 \wedge \text{isPaid} = \text{false})$ $\vee (\text{paymentAmount} > 62.5 \wedge \text{amount} > 35.0 \wedge \text{isPaid} = \text{false})$
③	Payment	Add penalty	$\text{amount} \leq 35.0 \wedge \text{isPaid} = \text{false}$
④	Send Appeal	End	$(\text{expense} \leq 15.6 \wedge \text{amount} > 33.6)$ $\vee (\text{expense} > 15.6 \wedge \text{dismissal} = "#")$ $\vee (\text{expense} > 16.0 \wedge \text{expense} \leq 19.85 \wedge \text{dismissal} = "\text{NIL}")$
⑤	Send Appeal	Receive Result	$(\text{expense} \leq 15.6 \wedge \text{amount} \leq 33.6)$ $\vee (\text{expense} > 15.6 \wedge \text{dismissal} = "$")$ $\vee (\text{expense} > 15.6 \wedge \text{expense} \leq 16.0 \wedge \text{dismissal} = "\text{NIL}")$ $\vee (\text{expense} > 19.85 \wedge \text{dismissal} = "\text{NIL}")$

Decision rule ①. The guard expression for the binding ① ([Notification]) of the activity *Send Fine* suggests that the fine processing continues only for cases in which the offender paid less than 30 Euros before the fine was sent. Indeed, we know that some fines are paid directly before the fine notification is received by the offender. The rule suggest that this happens mainly for inexpensive fines.

Decision rules ② and ③. The second guard expression was found for the binding ② of the activity *Payment*. The binding includes only the artificial end activity.

The rule indicates that the fine processing may finish for fines that are fully paid. However, the fine process may also finish for cases with a fine amount above 35 Euros and a low *paymentAmount* (i. e., the amount last paid) of below 18.78 or a high *paymentAmount* of above 62.5 Euros. We manually inspected the event log and found that, indeed, there are a few thousand cases in which the processing ends with unpaid fines in that range. In all of these cases a penalty had been applied. The guard for output binding © shows that *Payment* is followed by the *Add penalty* activity for cases with a low fine amount and in which the full fine amount was not yet paid.

Decision rules ④ and ⑤. The fourth and the fifth guard expressions for bindings ④ and ⑤ are both attached to the activity *Send Appeal*. The guard expressions are similarly to the decision rule discovered for transitions *Receive Result* and *INv6* (which directly leads to the end of the process) in Section 12.3. It seems to depend on the *expense* attribute whether the result of the appeal was received. However, the rules regarding ④ and ⑤ are more fine-grained, which is due to different settings when using C4.5.

Quality of the Discovered DC-Net

We checked the fitness and the precision scores of the discovered DC-Net by applying our multi-perspective conformance checking methods.⁵⁵ The fitness score is 0.95, which indicates that most of the observed behavior can be successfully replayed on the DC-Net. The multi-perspective precision score of the DC-Net with regard to the event log and the attributes selected for guard discovery is 0.69. Figure 12.10 presents the conformance diagnostics that are returned by the iDHM tool. From the more detailed conformance diagnostics, it can be seen that some of the bindings are grayed out. This indicates that these bindings are never activated when aligning the event log to the DC-Net, i. e., the routing transitions added the DPN for each binding of the DC-Net (see Section 8.4) are never executed. For example, the binding ④ for which a guard expression was discovered is actually never used. There are some more local conformance issues. For example, more than half of the occurrences of the activities *Appeal to Prefecture* and *Send Appeal* are only observed in the log (log move) but cannot be matched to the model behavior. Thus, even though the DC-Net is overall a good representation there are some local conformance issue. The good overall measures can be explained by the relative infrequency of the activities and bindings with conformance issues.

12.4.3 Comparison with State-of-the-art Techniques

We compared the result with two standard process discovery techniques:

⁵⁵We transformed the DC-Net into a DPN as described in Section 8.4 to apply our proposed methods.

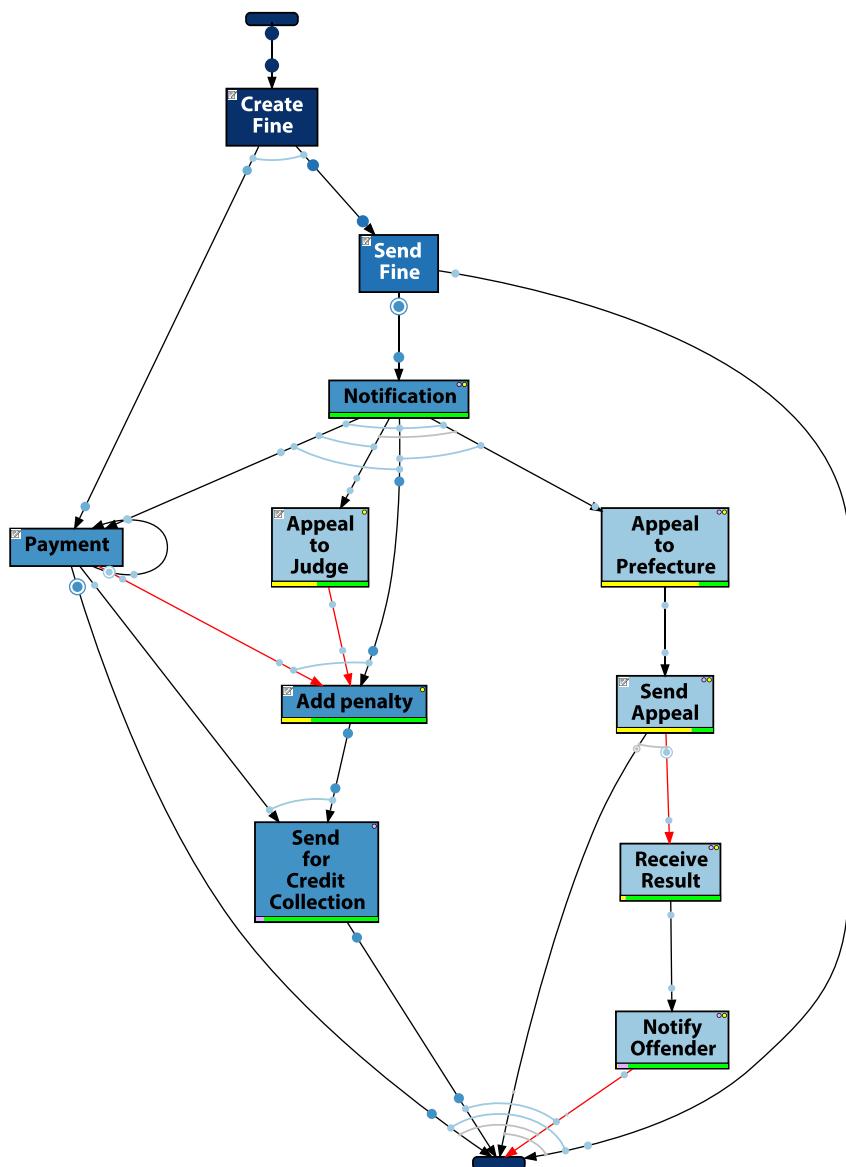


Figure 12.10: Conformance information projected on the discovered DC-Net. The color of the activity indicates its frequency and the color bar at the bottom of activities indicates the number of deviations in terms of model moves and log moves as described in Section 11.1.

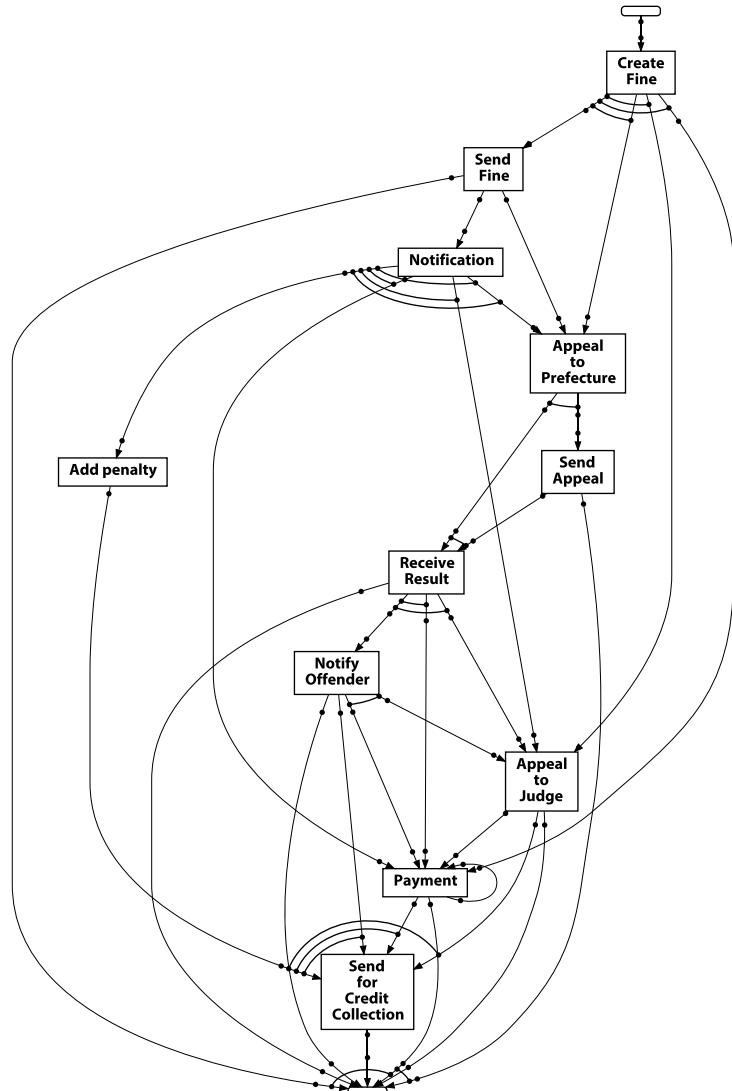


Figure 12.11: C-Net discovered by the standard Heuristic Miner for the fine management log.

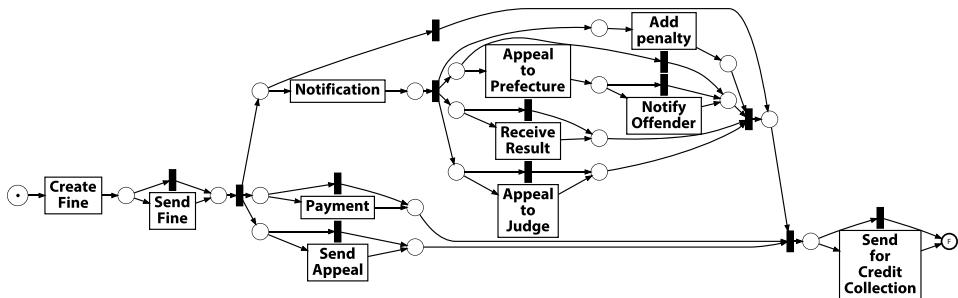


Figure 12.12: Petri net discovered by the Inductive Miner for the fine management log.

- the Heuristics Miner⁵⁶ (Figure 12.11) and
- the Inductive Miner⁵⁷ (Figure 12.12).

The C-Net returned by the Heuristics Miner contains many more dependencies than the DC-Net in Figure 12.9. It is very likely that some of these dependencies are the result of noise in the event log. For example, the dependency from *Appeal to Prefecture* to *Receive Result* is likely to be random noise since the result of the appeal can only be receive if the appeal has been sent beforehand. Moreover, the dependency from *Payment* to *Add penalty*, which was identified as conditional dependency ② in Figure 12.9 is not shown in Figure 12.11.

In the model obtained by the Inductive Miner none of the four relations added by our DHM method (①-⑤) are present. Moreover, applying decision mining techniques on the model discovered by the Inductive Miner will not provide the rules discovered by the iDHM since the necessary decision points are missing. For example, in Figure 12.12 there is no place that models the direct relation from *Payment* to *Add Penalty*. Thus, the conditional dependency ① would not be discovered on that model.

Clearly, it is possible to use different parameter settings for both methods. We tried several parameter settings and could not obtain a better result for both the Inductive Miner and the Heuristics Miner. The Inductive Miner returns a very imprecise model upon reducing the noise filtering parameter. Conversely, when increasing the noise filtering still none of the four relations is included in the model. Regarding the Heuristics Miner, it was possible to remove some of the dependencies that we consider as noise and retrieve a model similar to Figure 12.9 by increasing the θ_{obs} -parameter. However, already for a slight increase to 0.005 only the conditional dependency ⑤ remained in the discovered model. The remaining conditional dependencies were removed by the noise filtering of the Heuristics

⁵⁶We simulated the standard Heuristics Miner using the iDHM without considering the conditional dependencies and set $\theta_{obs} = 0$, which is the default setting of the Heuristics Miner.

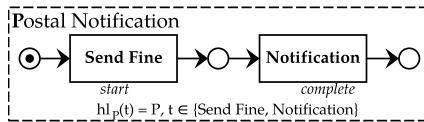
⁵⁷We used the Inductive Miner infrequent with the standard noise filtering parameter, i. e., 0.2.

Miner.

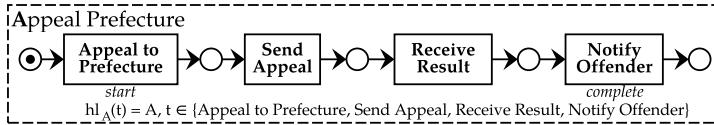
12.5 Guided Process Discovery

Finally, we applied the GPD method (cf., Chapter 9) to the fine management event log. We tested whether using some domain knowledge on the functional perspective of the process improves the process model discovered by state-of-the-art process discovery techniques.

12.5.1 Activity Patterns



(a) Activity pattern modeling the high-level activity *Postal Notification*.



(b) Activity pattern modeling the high-level activity *Appeal Prefecture*.



(c) Abstraction model composing both patterns in parallel.

Figure 12.13: The atomic activity patterns and the composed abstraction model used as input to the GPD method. Both atomic activity patterns were designed based on domain knowledge about the process.

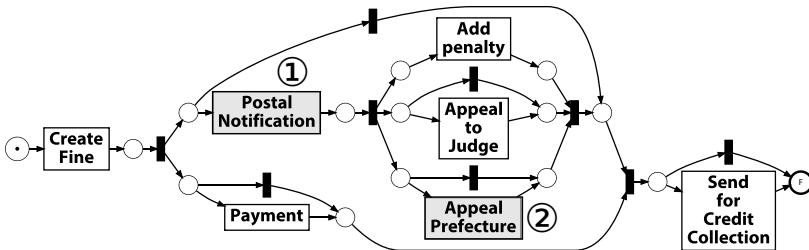
The GPD method requires a set of activity patterns as input. We created two activity patterns based on our domain knowledge about the process that we obtained from the normative process model in Figure 12.1. The activity patterns capture the behavior of two high-level activities:

1. appealing against the fine to the prefecture denoted as *Postal Notification* (**P**), and
2. sending and handling the postal fine notification denoted as *Appeal Prefecture* (**A**).

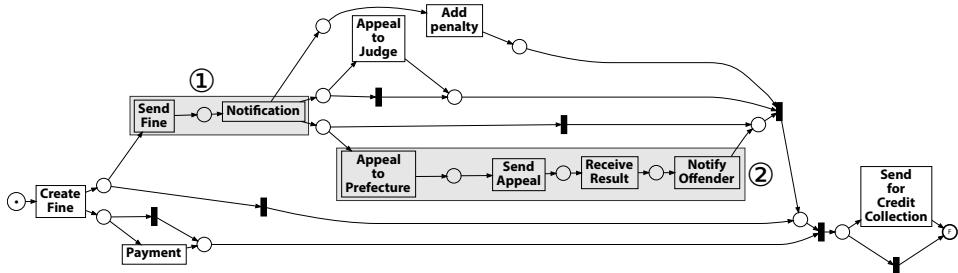
Figure 12.13 shows the DPN implementation of both activity patterns. We composed both patterns to the abstraction model $(P^{[0,\infty]} \diamond A^{[0,\infty]})$ (cf., Figure 12.13c),

i. e., we do not provide more domain knowledge: both high-level activities may be executed in parallel and be repeated indefinitely.

12.5.2 Discovery Results



(a) High-level process model discovered when applying GPD.



(b) Expanded process model in which high-level activities have been replaced by the activity patterns.

Figure 12.14: Process models discovered by the Inductive Miner when applying GPD.

We discovered a process model by applying the GPD method together with the Inductive Miner. Figure 12.14 shows the discovered process model. We first obtained an high-level event log from low-level events that are captured by the two activity patterns: *Postal Notification* and *Appeal Prefecture*. The *matching error* (cf., Section 8) was 0 for the high-level activity *Notification* and 0.04 for the high-level activity *Appeal Prefecture*, i. e., the activity patterns are of high quality. The *global matching error* was 0.247, i. e., 24.7% of the events could not be captured by the two activity patterns used as input (cf., line 8). We investigated the root cause and inspected the alignment that was computed as by the GPD method. There are several traces in which the an event recorded the activity *Send Fine* occurs without a subsequent event recording the activity *Notification*. Thus, the *Send Fine* event are diagnoses as log moves. Note that the choice whether to diagnose *Send Fine* as log move or to insert an model move with transition *Notification* depends on the cost function employed. Here, we preferred log moves over model moves.

Some events in original event already correspond to executions of high-level activities (i. e., activities *Create Fine*, *Payment*, *Appeal to Judge*, *Add penalty*, and *Send for Credit Collection*). Since the unmapped events are relevant for the process, they were added to the high-level event log in a post-processing step.

Then, we discovered the high-level process model shown in Figure 12.14a by applying the Inductive Miner on the resulting high-level event log. We expanded the two high-level activities in Figure 12.14a that we defined as input for the GPD method with the Petri nets of the corresponding activity patterns. Figure 12.14b shows the resulting expanded process model that is using the original low-level activities. As described in our GPD method (cf., Section 9.2.8). we can use this expanded model for conformance checking with regard to the original event log.

The main flow of the normative model that was shown in Figure 12.1 is reflected in the expanded model in Figure 12.14b, i. e., the sequence *Create Fine*, *Send Fine*, and *Notification* as well as the choice between two options to appeal against the fine. Thus, it may be easier to understand for stakeholders. However, the expanded process model excludes some of the process variants (e. g., the sequence *Create Fine* and *Send Fine* without further processing). Still, the fitness score of the expanded model is 0.81, which is not much lower than the fitness score 0.96 of the normative model. This indicates that these variants are infrequently observed.

12.5.3 Comparison with State-of-the-art Methods

We also compared the process model that was discovered by the Inductive Miner⁵⁸ on the original event log (denoted as *low-level event log* in the remainder of this section) with the expanded version of the process model that was discovered by applying the GPD method. The process model discovered by the Inductive Miner on the low-level event log was already shown in Figure 12.12 (cf., Section 12.4). This process models fits the event log better (fitness 0.99) compared to the expanded process model provided by our GPD method (fitness 0.81). However, it is not very precise (precision 0.61) since most activities can be skipped and executed in parallel. Again, the expanded process model is a more precise representation of the recorded events (precision 0.75). Thus, the process model discovered by the GPD method strikes a better balance between fitness and precision. Moreover, the semantically related activities *Appeal to Prefecture*, *Send Appeal*, *Receive Result*, and *Notify Offender* are not presented as such in Figure 12.12. For example, *Send Appeal* and *Appeal to Prefecture* are placed in different parts of the process model and may be executed in parallel. Conversely, the expanded process model in Figure 12.14b semantically groups related activities together and provides a view on the function perspective of the process. We exploit the hierarchical structure that was imposed by using the activity patterns.

⁵⁸Again, we used the Inductive Miner infrequent with its standard noise filtering parameter, i. e., 0.2.

12.6 Conclusion

We applied *all of our proposed methods* in the context of the fine management process for which we obtained a real-life event log and a normative process model.

- In Section 12.2, the analysis of the normative process model with our proposed *balanced alignment* conformance checking technique provided valuable insights regarding the adherence of the process to regulations, which would not have been returned by earlier multi-perspective alignment techniques. For example, the approach discussed in [LA13a] returns futile results in a number of cases, such as diagnosing non-plausible deviations on the amount of the fines. The insights obtained show that unpaid fines should be monitored more closely.
- In Section 12.3, the decision mining method discovered several guards. The guards discovered for one of the decision points were overlapping and found to be plausible. Moreover, the discovered overlapping guards helped to balance fitness and precision of the discovered data-aware model.
- In Section 12.4, we showed that several infrequent process paths associated with deterministic data-based rules can be revealed by using the DHM method. Each of the discovered conditional dependencies could be interpreted in the context of the fine management process and provided new insights into the actual process behavior.
- In Section 12.5, we used the GPD method to abstract several activities to higher-level activities. We used the function perspective of the process to improve the process model discovered by the Inductive Miner. The model discovered by the GPD method strikes a better balance between precision and fitness than the model discovered by Inductive Miner on the original event log.

Overall, all proposed methods could be successfully applied and provided actionable insights that may be used to improve the fine process.

13 Case Study: Sepsis

In our second case study, we applied the methods proposed in this thesis to a real-life event log that we obtained from an Enterprise Resource Planning (ERP) system⁵⁹ of a regional hospital in The Netherlands. Parts of this case study were published in [Man+16b; MB17].

This chapter is structured as follows. First, we describe the context of the case, specific process questions, the extracted event log, and a hand-made normative process model in Section 13.1. Then, we present the application of our multi-perspective conformance checking method on the normative process model in Section 13.2 followed by an evaluation of the proposed decision mining method in Section 13.3. Then, we describe how we applied both our DHM method (Section 13.4) and our GPD (Section 13.5) method to automatically discover process models without using the normative process model.

13.1 Case Description

We conducted the *sepsis case study* in a regional hospital in The Netherlands. The regional hospital has about 700 beds at several locations and is visited by about 50,000 patients per year. The scope of our case study was on the *trajectories* of patients that are admitted to the *emergency ward* of the hospital. From the side of the hospital, the head doctor and a nurse from the emergency department as well as a data analyst from the business intelligence department of the hospital were involved. The medical staff provided medical domain knowledge, process questions, and validated our results, whereas the data analyst helped with the data provision.

We focused on a *specific sub group of patients*: those that were suspected of having a sepsis. As illustrated in Figure 13.1, sepsis is a life-threatening condition that requires immediate treatment [Rho+17]. We considered all patients that displayed any symptom that may be related to sepsis, i. e., all patients that were more closely screened in the emergency ward of the hospital. Typical symptoms of a sepsis are:

- increase body temperature,
- increased heart rate,

⁵⁹The system used is based in SAP and customized for the hospital.

⁶⁰Retrieved on 21/8/2017 from <http://survivingsepsis.org/SiteCollectionDocuments/Surviving-Sepsis-Campaign-2016-Guidelines-Presentation-Final.pptx>

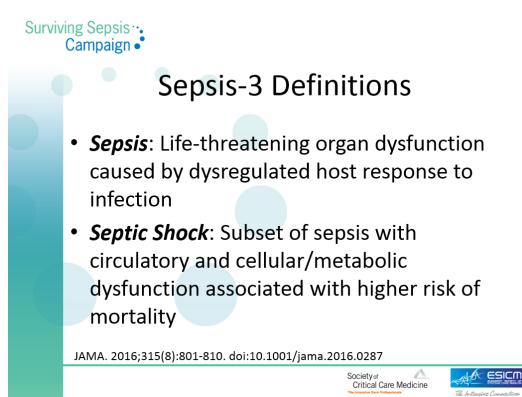


Figure 13.1: Definition of sepsis and septic shock as provided by the Surviving Sepsis Campaign⁶⁰. All of the patients considered in the case study have been screened for sepsis in the emergency ward of the hospital.

- high respiratory rate, and
- abnormal white blood cell count.

Together these symptoms are known as Systemic Inflammatory Response Syndrome (SIRS) criteria, which have been used as indications for sepsis.⁶¹ Patients with multiple SIRS criteria are of particular interest for the hospital since patients that develop a sepsis are associated with a high mortality risk and patients with septic shock with an even higher risk. Thus, we focused on a single group of patients for which a specific treatment is to be expected. By doing so, we aimed to avoid some of the complexity and flexibility associated with health care processes [RF12].

13.1.1 Process Questions

Figure 13.2 shows an illustration of the patient flow that was created by process stakeholders from the emergency department (i.e., a nurse and the head doctor). This document served as starting point for the process mining analysis. It helped to clarify the assumptions and perspectives on the process of nurses and doctors in the emergency ward. Based on Figure 13.2, we identified several question and analysis tasks together with all stakeholders from the hospital:

1. Are particular medical guidelines (i.e., the guideline in [Rho+17]) for the treatment of sepsis patients followed? More specifically:

⁶¹More recently, after we conducted the case study, the SIRS criteria are not recommended as sole criteria for sepsis anymore by the current guidelines. Amongst other because they are overly sensitive [M+16].

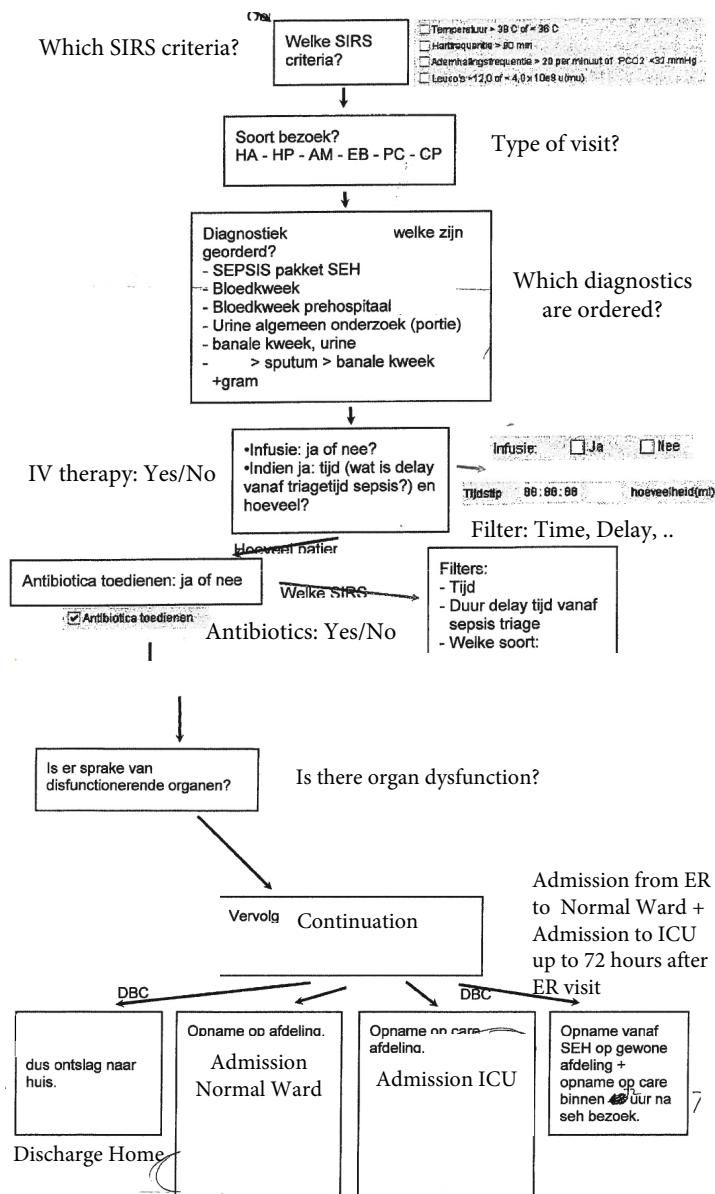


Figure 13.2: Patient flow and questions as described by the process stakeholders.

Table 13.1: Activities recorded in the sepsis cases event log.

Activity	Frequency	Description
Leucocytes	3,383	Inserting the measurement of the white blood-cell count into the system.
CRP	3,262	Inserting the measurement of the C-reactive protein level into the system.
LacticAcid	1,466	Inserting the measurement of the lactic acid level into the system.
Admission NC	1,182	Admission of the patient to a normal care ward.
ER Triage	1,053	Filling of the general triage document finished.
ER Registration	1,050	Initial registration of the patient in the emergency ward.
ER Sepsis Triage	1,049	Filling of the specific sepsis triage document finished.
IV Antibiotics	823	Administering of antibiotics via intravenous therapy.
IV Liquid	753	Administering of liquid via intravenous therapy.
Release A	671	Discharge of the patient with code A.
Return ER	294	Return of the patient to the emergency ward.
Admission IC	117	Admission of the patient to a intensive care ward.
Release B	56	Discharge of the patient with code B.
Release C	25	Discharge of the patient with code C.
Release D	24	Discharge of the patient with code D.
Release E	6	Discharge of the patient with code E.

- Are patients administered antibiotics within one hour⁶² and
 - are lactic acid measurements done within three hours?
2. Visualize and investigate the following specific trajectories:
- discharge without admission,
 - admission to the normal care,
 - admission to the intensive care, and
 - admission to the normal care and transfer to intensive care.
3. What are the trajectories of patients that return within 28 days since these patients are assumed to be problematic for the hospital.

We will use these questions to guide the application of our proposed multi-perspective process mining techniques and to show that they can address relevant questions in real-life cases.

13.1.2 Event Log

Based on the document shown in Figure 13.2 and together with stakeholders from the hospital, we could identify several sources for event data:

- the triage document filled in for sepsis patients with information on
 - the time the triage was conducted,
 - the symptoms present (SIRS criteria for sepsis),

⁶²Here, the hospital aims to exceed the three hour recommendation by the guideline in [Rho+17]

Table 13.2: Attributes recorded in the sepsis cases event log.

Attribute	Domain	Description
Age	discrete	Age of the patient grouped in 5-year intervals and a single group for patients of age 90 years or older.)
CRP, Leukocytes, Lactic Acid	continuous	Blood sample measurement results
Diagnose	literal	Code for the diagnosis
DiagnosticArtAstrup,	boolean	Various checkboxes that can be ticked on the triage document.
DiagnosticBlood, ...	boolean	
DisfuncOrg	boolean	Checkbox on the triage document indicating a dysfunctional organ.
Hypotensie	boolean	Checkbox on the triage document indicating hypotension, i.e., low blood pressure.
Hypoxie	boolean	Checkbox on the triage document indicating hypoxia, i.e., low oxygen supply.
InfectionSuspected	boolean	Checkbox on the triage document indicating whether an infection is suspected.
Infusion	boolean	Checkbox on the triage document indicating whether intravenous infusions are required.
Oligurie	boolean	Checkbox on the triage document indicating a reduced output of urine (Oliguria).
SIRSCritHeartRate	boolean	Checkbox on the triage document indicating an elevated heart rate (above 90 beats per minute).
SIRSCritLeucos	boolean	Checkbox on the triage document indicating a low white blood cell count.
SIRSCritTachypnea	boolean	Checkbox on the triage document indicating a high respiratory rate (Tachypnea).
SIRSCritTemperature	boolean	Checkbox on the triage document indicating an elevated temperature (less than 36°C or greater than 38°C).
SIRSCriteria2OrMore	boolean	Flag that indicates whether two or more of the SIRS criteria checkboxes have been ticked.

- the diagnostics ordered, and
- the time infusions of liquid and antibiotics were administered;
- documents created by the laboratory for several blood tests; and
- information about the further trajectory of the patient recorded by financial systems.

All documents and records are stored in different databases of the ERP system that is used by the hospital. We collected data about several activities that are executed for this group of patients from three different information systems of the hospital.

The activities can be coarsely categorized into *medical or treatment activities* and *logistical or organizational activities* [LR07]. We collected the event data from several sources and consolidated it into a single anonymized event log covering the traces that were recorded for 1,050 patients over the course of 1.5 years in the hospital's ERP system. The resulting event log was made available for further process mining research purposes as part of the collection of real-life event logs of the IEEE Task

Force on Process Mining [Man16a].⁶³

The event log contains events for 16 activities, which are listed together with their frequencies in Table 13.1. Most activities captured in that event log are based on organizational activities, e.g., the discharge of the patient or the registration in the emergency ward. However, we also include some patient treatment related events such as the filling the triage document with information on the condition of the patient and the measurements of some typical indicators for a sepsis: leukocytes, CRP, and lactic acid.

Next to the information about the execution of activities, the event log contains several data attributes, all of which are listed in Table 13.2. Most of the data attributes originate from a sepsis triage form, which is usually filled at the start of the process. On this form, the medical staff (i.e., nurses) enter data about the condition of the patient. The triage form is organized in the form of a checklist.⁶⁴ The blood measurement attributes CRP, Leukocytes, and Lactic Acid are updated several times throughout the process.

13.1.3 Normative Process Model

Together with the data analyst from the hospital, we iteratively created a normative process model that can be used to visualize the patient trajectories and to check the conformance to some of the medical guidelines. The normative model in the DPN notation is shown in Figure 13.3. We use two guard expressions and two variables to encode the two constraints on the time perspective of the medical guideline for sepsis patients. According to the international sepsis guidelines [Rho+17], patients should be administered antibiotics within one hour (i.e., 60 minutes) and the lactic acid measurement should be finished within three hours (i.e., 180 minutes). Variable *timeTriage* records the time at which the activity *ER Sepsis Triage* was executed, i.e., it is only written by that activity. Variable *timeAntibiotics* records the time at which activity *IV Antibiotics* is executed and variable *timeLacticAcid* records the time at which activity *LacticAcid* is executed. The variables record the time in minutes since the start of the case, i.e., the time relative to the start of the case. Table 13.3 lists the guard expressions of the guarded transition in Figure 13.3.

Table 13.3: Guard expressions encoding the time constraints of the sepsis process.

Activity	Guard Expression
LacticAcid	$\text{timeLacticAcid}' \leq \text{timeTriage} + 180\text{min}$
IV Antibiotics	$\text{timeAntibiotics}' \leq \text{timeTriage} + 60\text{min}$

⁶³The sepsis cases event log can be obtained from: <https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460>

⁶⁴For confidentiality reasons we cannot disclose the underlying checklist.

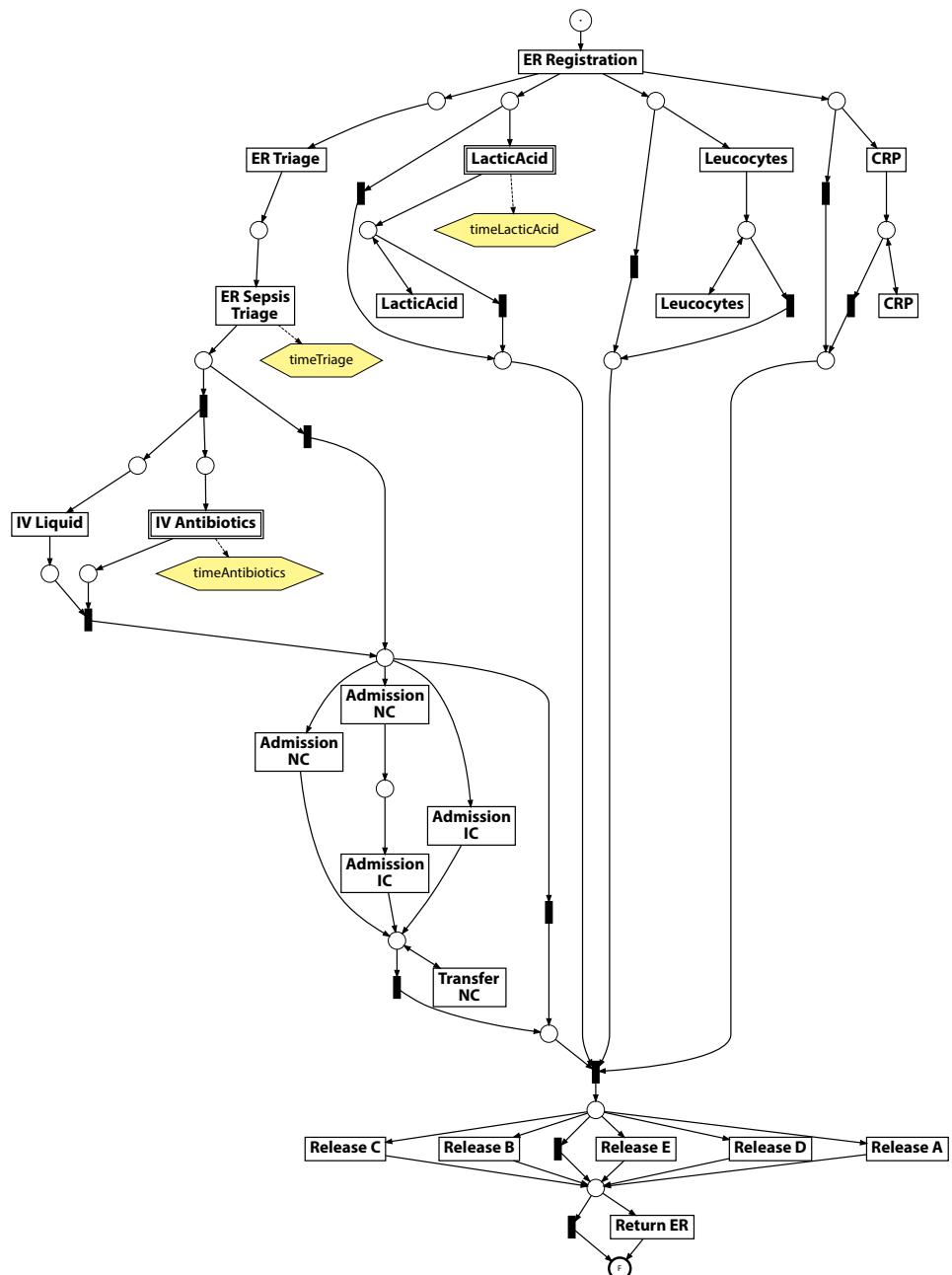


Figure 13.3: Hand-made normative process model for the trajectories of sepsis patients.

13.2 Conformance Checking

The normative process model encodes rules from the medical guideline with regard to sepsis patients. We checked the overall conformance between event log and the normative process model using a multi-perspective alignment (cf., Section 5.2). Thus, we focused on the first two process questions regarding the administration of antibiotics and the measurement of lactic acid.

13.2.1 Configuration Settings and Cost Function

We enriched each event of the sepsis event log with an additional attribute *time*, which records the elapsed time since the start of the trace in minutes. We mapped all three DPN variables *timeTriage*, *timeAntibiotics*, and *timeLacticAcid* to the attribute *time*, e. g., $v(\text{timeTriage}) = \text{time}$ (cf., Definition 3.12 for the variable label function v). The transitions of the DPN are mapped to activities in the event log with the same name with the exception of the transition *Transfer NC*, which is mapped to the activity *Admission NC*, i. e., $\lambda(\text{TransferNC}) = \text{AdmissionNC}$ (cf., Definition 3.12 for the activity label function λ). Regarding the cost function, we use a cost function that assigns high cost to log moves (cost 10 each), medium cost to model moves (cost 5 each), and low cost to incorrect synchronous moves (cost 1 each). This encodes our assumption that a violation of the time perspective rules is more likely than a violation of the control-flow of the process.⁶⁵ Moreover, we want to reduce the number of log moves introduced by the alignment. The assumption is that the process model is correct and it is more likely that an event has not been recorded. Here, the cost function does not directly reflect the severity of violations from a domain viewpoint but it is used to steer the alignment towards the most likely alignment.

13.2.2 Conformance Checking Results

Figure 13.4 shows the resulting projection of conformance information on the process model. The rule regarding administering antibiotics within one hour is sometimes violated: *IV Antibiotics* and *timeAntibiotics* in Figure 13.4 are colored orange. The average time between *ER Sepsis Triage* and *IV Antibiotics* is 1.7 hours and the rule is violated for 58.5% of the cases. The situation regarding the lactic acid measurement rule is much better. This rule, which encodes the timely measurement of lactic acid within three hours, is only violated in 0.7% of the cases.

The high number of violations regarding the antibiotics rule can be explained by two factors:

⁶⁵Additionally, we set the uninitialized variable mode to *FREE* since it might happen that activity *LacticAcid* occurs before the variables *timeTriage* has been written. This should not be considered as deviation.

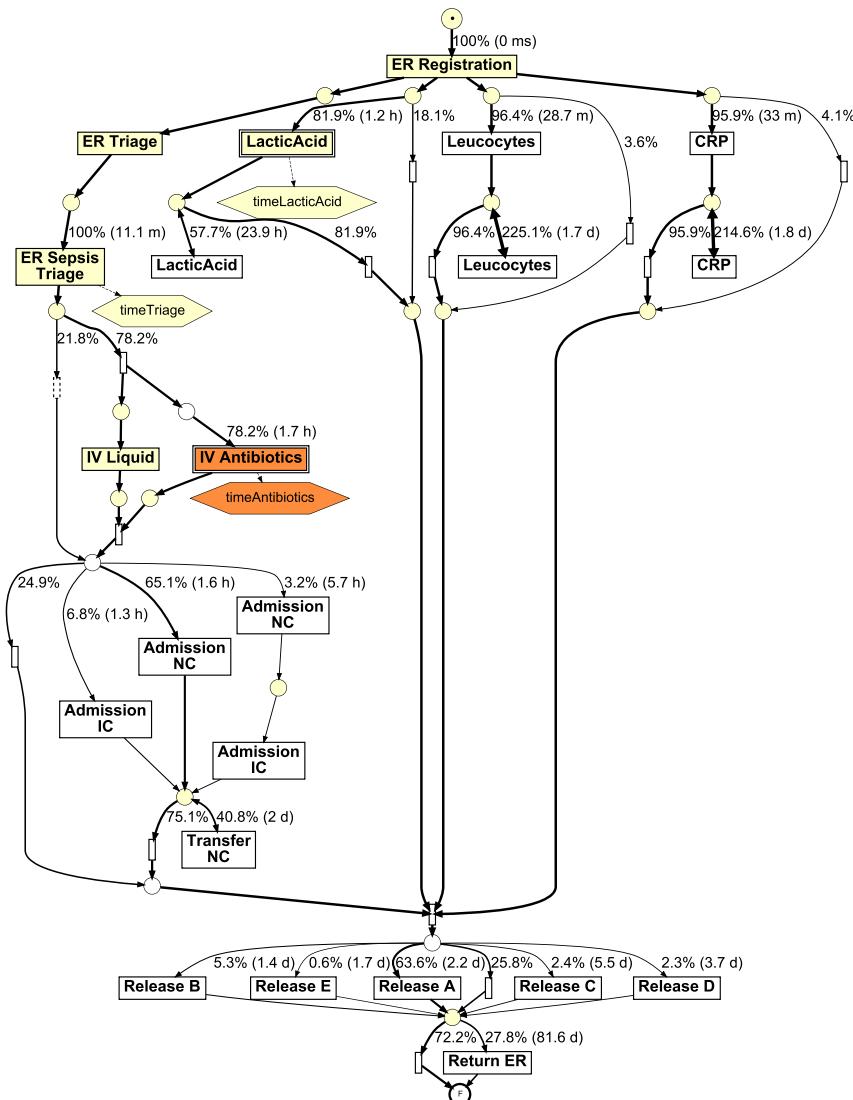


Figure 13.4: Projection of the events on the normative process model as returned by the MPE. There are violations regarding the variable *timeAntibiotics* are visible (highlighted in orange) and some violations regarding variable *timeLacticAcid*. For some of the transition a few model moves are diagnosed (highlighted in yellow).

2. Behandeling		Kies!	Datum
Infusie:	<input type="checkbox"/> Ja	<input type="checkbox"/> Nee	
Vochtresuscitatie	NaCl	25.11.2013	Tijdstip 11:01:49
<input checked="" type="checkbox"/> Antibiotica toedienen	250	25.11.2013	Tijdstip 11:01:50
	500		<input type="checkbox"/> Geen Antibiotica toegediend
	1000		

Figure 13.5: Digital triage form that is filled out in the emergency ward.

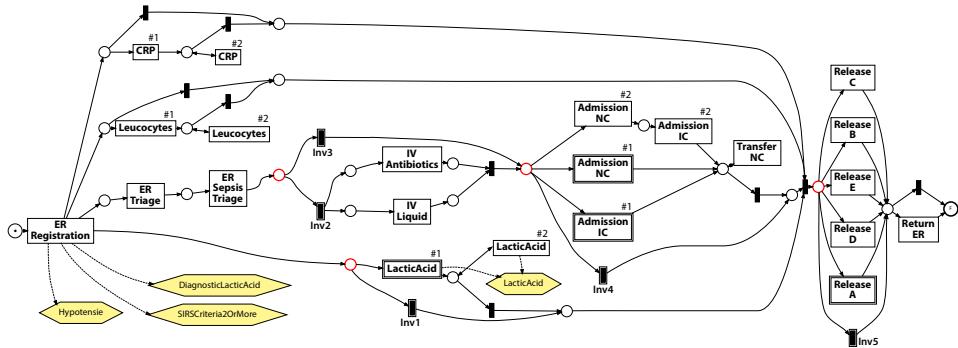
1. Some guidelines recommend administering antibiotics within one hour [Rho+17]; however, it was acknowledged that this is not always feasible. Moreover, not all patients in our event log show sufficient symptoms of a sepsis. Thus, the one hour rule followed by the hospital is rather strict.
2. The data about the infusion of antibiotics is entered manually by nurses and doctors in a form as shown in Figure 13.5. When looking at the data, we found some cases in which the entered antibiotics timestamp is 24 hours after the triage and other cases in which the antibiotics timestamp was before the triage document was filled in. Thus, it is difficult to interpret these measurements since some of the diagnostics may result from poor data quality.

Still, the application of conformance checking was considered useful by the data analyst and head doctor of the emergency ward. It revealed an underlying data quality issue and a doctor found the analysis to be “a magnificent way to clarify patient flow”.

13.3 Discovery of the Data Perspective

Next, we checked whether additional rules can be discovered without decision mining technique (cf., Chapter 10). As input to the MPE data-discovery mode, we used a Petri net version of the DPN shown in Figure 13.3, i. e., all existing guards and variable write operations were removed. We set the parameters to non-binary rules and a minimum instances ratio of 0.1. Figure 13.6 shows the decision rules that were discovered by the overlapping decision mining technique. The discovered guards are defined using four variables: *Hypotensie*, *DiagnosticLacticAcid*, *SIRSCriteria2OrMore*, and *LacticAcid* (cf., Table 13.2).

The first two guards in Figure 13.6b suggest that the execution of the activity *LacticAcid* is determined by the value of *DiagnosticLacticAcid*. Indeed, we confirmed that this flag indicates that the lactic acid level should be conducted as measured for that specific patient. The next two guards concern the invisible routing transitions *Inv2* and *Inv3*. *Inv2* is executed if the checkbox *SIRSCriteria2OrMore* was ticked (i. e., *SIRSCriteria2OrMore* = true). Executing *Inv2* in turn requires that both liquid and antibiotics are administered via intravenous therapy. A doctor from the hospital



(a) DPN discovered by the overlapping decision mining method. We highlighted the places for which we discovered decision rules in red.

Transition	Guard expression
Inv1	DiagnosticLacticAcid = false
LacticAcid	DiagnosticLacticAcid = true
Inv2	SIRSCriteria2OrMore = true
Inv3	SIRSCriteria2OrMore = false
Admission NC	LacticAcid > 0
Admission IC	LacticAcid > 0 ∧ Hypotensie = true
Inv4	(LacticAcid > 0 ∧ Hypotensie = false) ∨ LacticAcid ≤ 0
Inv5	LacticAcid ≤ 0
Release A	LacticAcid > 0

(b) Guards discovered by the overlapping decision mining method.

Figure 13.6: DPN discovered by the overlapping decision mining method when applied on the sepsis cases event log and the normative process model without guards.

confirmed that those patients should, indeed, be administered antibiotics and liquid. These guards were discovered by standard decision mining techniques [LA13b] as well since the expressions are mutually exclusive.

We also discovered a decision rule for the decision point that determines the type of admission: normal care (*Admission NC*), intensive care (*Admission IC*), no admission to the hospital (*Inv4*), or the problematic trajectory, i. e., first an admission to normal care and, then, an admission to the intensive care. For this decision point, our overlapping decision mining method discovered the overlapping decision rule listed in Figure 13.6b. Patients with a lactic acid measurement ($\text{LacticAcid} > 0$) can always be admitted to normal care. As an alternative to normal care, if attribute *Hypotensie* = true then patients can also be admitted to intensive care; otherwise, if *Hypotensie* = false patients leave the hospital. The guards for the activities overlap and the final decision is likely to be made on the basis of contextual factors, which are not encoded in the event log. Note that we already discussed these guards in Section 10.4 as part of the method's evaluation. Standard decision mining methods

did not return a rule for transition *Admission IC*, which can be interpreted either as the always fulfilled guard true (i. e., a fully overlapping rule) or the never fulfilled guard false (i. e., a strict mutually-exclusive rule). Our method aims to strike the balance between these two extremes.

Finally, our approach discovered a second overlapping decision rule for the decision point that is concerned with the discharge of the patients. Unfortunately, guard expressions could be discovered only for two transitions: *Release A*, the most frequent discharge type, and *Inv5*, patients without any discharge information. All other transitions are always enabled, i. e., the guard expression true was discovered. Patients without a lactic acid measurement (i. e. $\text{LacticAcid} \leq 0$) are likely to be leaving the hospital without discharge information. From a more detailed analysis of these patients it could be seen that most of them have never been admitted to the hospital ward.

13.4 Data-aware Process Discovery

We also tried to discover a process model from scratch by applying our data-aware process discovery method to the event log. We used the iDHM (cf., Chapter 8 and Section 11.1) to discover a DC-Net that reveals both the main process behavior as well as infrequent data-dependent paths.

13.4.1 Configuration Settings

Figure 13.7 shows the DC-Net discovered by the iDHM for the sepsis cases event log. This took about 3 seconds. We used the following, experimentally determined, parameter settings for the iDHM:

- the *accepted-task-connected heuristic*, since some activities might not be of interest;
- the observation threshold $\theta_{obs} = 0.1$ (i. e., relations that appear in more than 10% of the log trace), because we want to capture the main flow of the process and there is a lot of variability in the event log;
- the dependency threshold $\theta_{dep} = 0.95$ to discover only very strong dependencies;
- the binding threshold of $\theta_{bin} = 0.001$ to exclude very infrequent bindings.

Moreover, we used all of the attributes that are available to discover dependency conditions and guarded bindings. We used C4.5 with 10-times 10-fold cross validation and only accepted classifications with a conditional dependency measure of $\theta_{con} \geq 0.7$ (cf., Section 8.3.1). Thus, only very good classifiers that generalize well were used to add conditional dependencies. We also used a standard C4.5 classifier to discover the binding guards. Here, we used the F1-score as an evaluation measure. We included those guards that obtain an F1-score of at least 0.8.

13.4.2 Discovery Results

The iDHM revealed two conditional dependencies that are highlighted with the color red and assigned numbers in Figure 13.7 (① and ②). Moreover, three bindings of the DC-Net are assigned a guard expression (Ⓐ, Ⓑ, and Ⓒ).

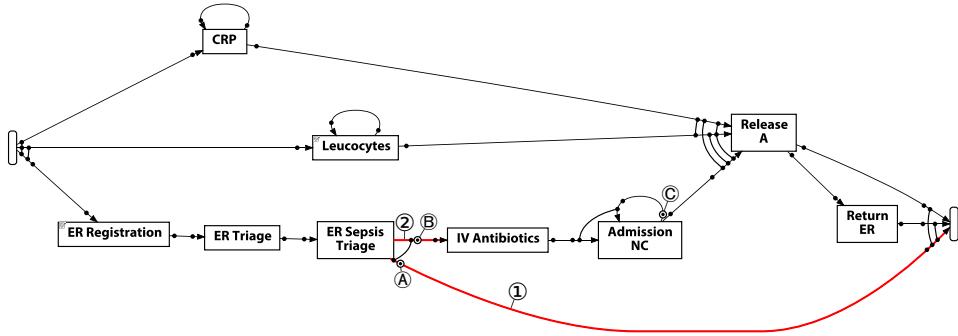


Figure 13.7: DC-Net discovered by applying the iDHM to the sepsis cases event log.

Conditional Dependencies

The two conditional dependencies both concern the activity *ER Sepsis Triage*. The first conditional dependency added (①) reveals that some patients leave the hospital after the sepsis triage form was filled in. The discovered dependency condition (*InfectionSuspected* = *false*) is almost perfect (Cohen's kappa value of 0.95) and was observed to be fulfilled in 47 cases. Indeed, based on our domain knowledge this dependency condition seems to be valid. Patients that are not suspected of having an infection are more likely to skip treatment under the sepsis protocol. However, further confirmation from the medical staff would be required. The second conditional dependency added (②) is associated with the negation of the rule for dependency ①, i. e., *InfectionSuspected* = *true*.

Guarded Bindings

Three guards (Ⓐ, Ⓑ, and Ⓒ) are discovered for output bindings of the activities *ER Sepsis Triage* and *Admission NC*. Interestingly, the decision rules for the bindings of *ER Sepsis Triage* (Ⓐ and Ⓑ) are based on the attribute *SIRSCriteria2OrMore* and not on the attribute *InfectionSuspected*. For a value of *true*, the patient remains in the hospital and for a value of *false*, the patient leaves the hospital. It might come as a surprise that the condition discovered for the bindings differs from the condition discovered for the dependency conditions. The explanation for the difference is as follows. For the discovery of decision rules for output bindings only events that

match the discovered C-Net structure are used whereas all events are used for the discovery of dependency conditions. Moreover, both attributes *SIRSCriteria2OrMore* and *InfectionSuspected* are likely to be correlated. Patients without an infection are expected to show less of the SIRS criteria. Note that our method did not discover a guard for the output binding {IVAntibiotics, End}. Indeed, this output binding does not make sense from a domain perspective since the *End* activity needs to be executed by all cases. The binding was discovered due the employed heuristic based on the Flexible Heuristics Miner [WR11].

Quality of the DC-Net

Figure 13.8 shows the result of applying the multi-perspective conformance checking feature of the iDHM. The optimal alignment between the DC-Net (i.e., the DPN representation of it) and the event log could be computed within 90 seconds. Generally, it shows that the activities that are included in the DC-Net can be aligned well to the event log. However, there is a considerable number of model moves diagnosed for activity *Release A*, which indicates that it does not always follow the activity *Admission NC*. In fact, in comparison with the normative process model (cf., Figure 13.3) all other *Release* activities are missing. This is because we focused the discovery on the main process flow and only includes infrequent behavior that can be characterized by deterministic rules over the process data.

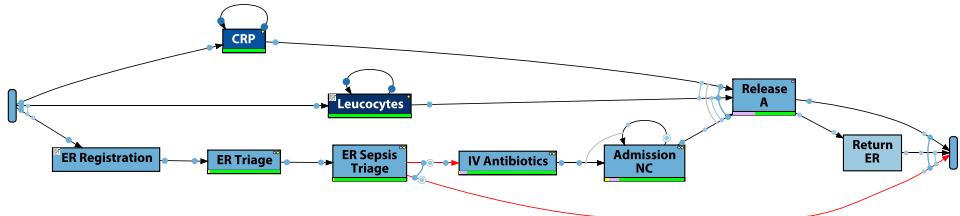


Figure 13.8: Conformance diagnostics obtained by a multi-perspective alignment. The iDHM projects the alignment information onto the DC-Net.

13.5 Guided Process Discovery

We used the GPD method on the event log of the sepsis patients to discover a process model that is close to the assumptions of the stakeholders of the process. We show that the process model obtained with the GPD is suitable to answer the process questions. Moreover, the discovered process model is comprehensible by stakeholder and, thus, it is more useful in the communication with them than process models discovered by state-of-the-art process discovery techniques. Note that in contrast to Section 13.2, in which we use the normative model to conduct

conformance checking, the GPD method only requires some limited knowledge on the process as input. Here, we do not assume complete knowledge of the desired process model.

We aim to use the discovered process model to address the following three specific process questions that are based on the more generic process questions were introduced in Section 13.1:

1. What are the trajectories of patients depending on how they were initially admitted to the hospital? Is there any influence on the remaining process, e.g., does a certain category of patient return more often. Specifically, the hospital is interested in the following three categories: (1) patients that are first admitted to the *normal care ward*, (2) first to the *intensive care ward*, or (3) patients that are first admitted to the *normal care ward* and, only then, to the *intensive care ward*? Each of the categories is of interest to the hospital because the location of the first admission indicates the severity of the sepsis condition. In particular, the third category is of high interest as it indicates that the patient's condition has worsened after being admitted. The hospital is interested in minimizing the number of these patients. Moreover, they want to visualize the effects of this category of patients to the operation of the hospital, e.g., in terms of the outcome and the time that those patients remain in the hospital.
2. Are patients with a sepsis condition given antibiotics and liquid and, if so, what is the delay from the initial triage activity until antibiotics are administered in the emergency ward? The rationale for this question is a medical guideline that recommends the administration of antibiotics within one hour after a sepsis condition has been identified.
3. Are there decision rules that can be discovered based on the attributes recorded in the event log? Discovering such rules may lead to insights on what kind of patients follow a certain trajectory in the process. It would be interesting for the hospital to know the most likely trajectory of a patient in order to prevent problems from arising early in the process.

First, we define a set of manually designed and discovered activity patterns in Section 13.5.1. Then, we apply our GPD method and discuss the results in Section 13.5.2. Finally, we show that state-of-the-art process discovery methods applied directly on the low-level event log provided process models that were unsuitable to answer these questions (Section 13.5.3).

13.5.1 Activity Patterns

We identified two manual and three discovered activity patterns. The manual patterns are shown in Figure 13.9a and Figure 13.9b. Both patterns are based on the

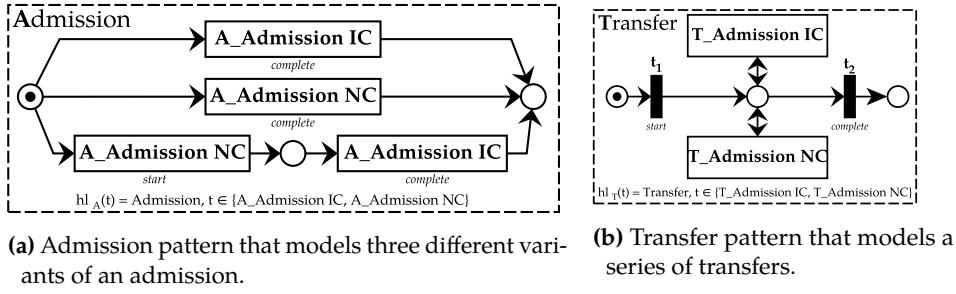


Figure 13.9: Two manual patterns that were created for the sepsis event log. The respective activity mapping is defined by removing the prefixes *A_* or *T_* from the transition name (e.g., $\lambda_A(A_{\text{Admission IC}}) = \text{Admission IC}$).

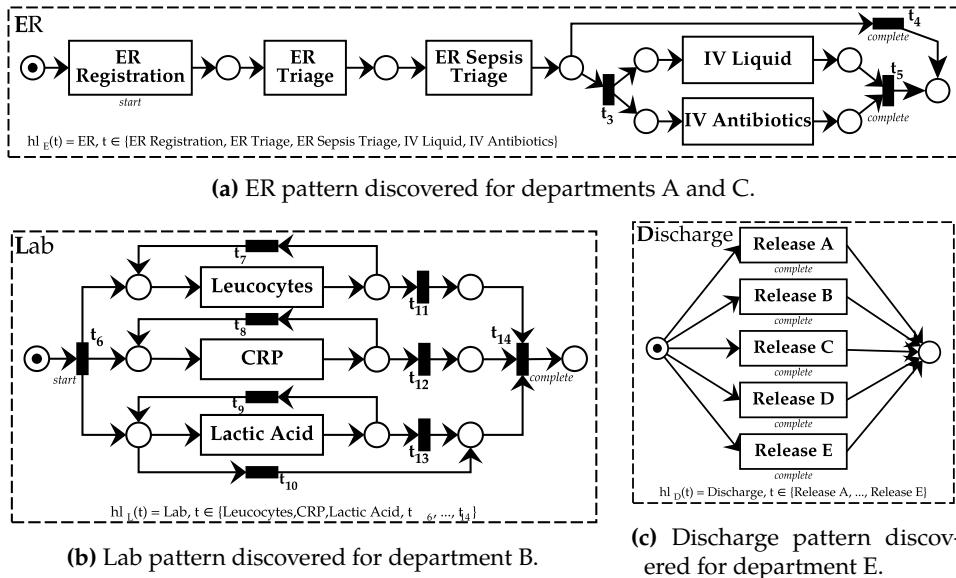


Figure 13.10: Three discovered patterns that were obtained by splitting the log based on the department attribute and using the Inductive Miner on the resulting sub logs.

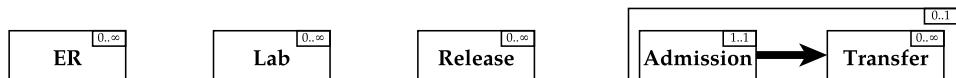


Figure 13.11: Abstraction model used for the case study. We added the restriction that the high-level activity Transfer can only occur after the high-level activity Admission.

activities *Admission NC* and *Admission IC*, which relate to the admission of a patient to an normal care or intensive care ward, respectively. We designed the manual patterns together with the data analyst of the hospital based on the first process question that was articulated by a doctor from the hospital. Activity pattern *Admission* (Figure 13.9a) encodes the three admission variants, which were also encoded in a similar manner in a flowchart-like process documentation provided by the emergency department (cf., Figure 13.2). We deliberately duplicated the activities to encode the problematic third variant of admission that is of great interest to the doctor. Since a patient may further be transferred between the different wards of the hospital, there may be further events that record one of the low-level admission activities. Subsequent transfers are not of interest to the emergency department. Therefore, we added activity pattern *Transfer* (Figure 13.9b), which encodes that any number of subsequent transfers may occur.

Moreover, we obtained three discovered patterns based on information on the organizational perspective that is part of the event log. Based on the department that executed the activity, we extracted three sub logs. Each log contained all activities performed by employees of a certain department. Then, we discovered a process model for each of the sub logs using the Inductive Miner (Figure 13.10). Activity pattern *ER* (Emergency Department) is shown in Figure 13.10a. All activities in pattern *ER* are executed in the emergency department (departments A and C in the event log). Therefore, we denote this pattern as *ER*. First, the patient is registered and triage checklists are filled. Then, antibiotics and liquid infusions can be given. Figure 13.10b shows activity pattern *Lab* discovered for department B, clearly this is the laboratory department responsible for blood tests. Figure 13.10c is based on a sub log obtained for department E and contains five different activities that relate to different (anonymized) variants on how patients are discharged.

We composed the five activity patterns into an abstraction model (Figure 13.11). We used the composition functions to add the constraint that a *Transfer* can only occur after an *Admission* has taken place. Clearly, patients need to be admitted before they can be transferred. Overall, we used only basic domain knowledge on the process and organizational information taken from the event log to build an abstraction model.

13.5.2 Discovery Results

We created an abstracted high-level event log with the abstraction model shown in Figure 13.11. The abstracted event log has about 8,300 events for the six high-level activities.⁶⁶ The abstracted event log could be computed in less than 2 minutes using 2 GB of memory. The global matching error of the abstraction model was $\epsilon = 0.02$. Only for pattern *ER* a non-zero local matching error $\epsilon(\text{ER}) = 0.006$ was recorded, i.e., all other patterns match perfectly.

⁶⁶We did not create an activity pattern for one activity: *Return ER*.

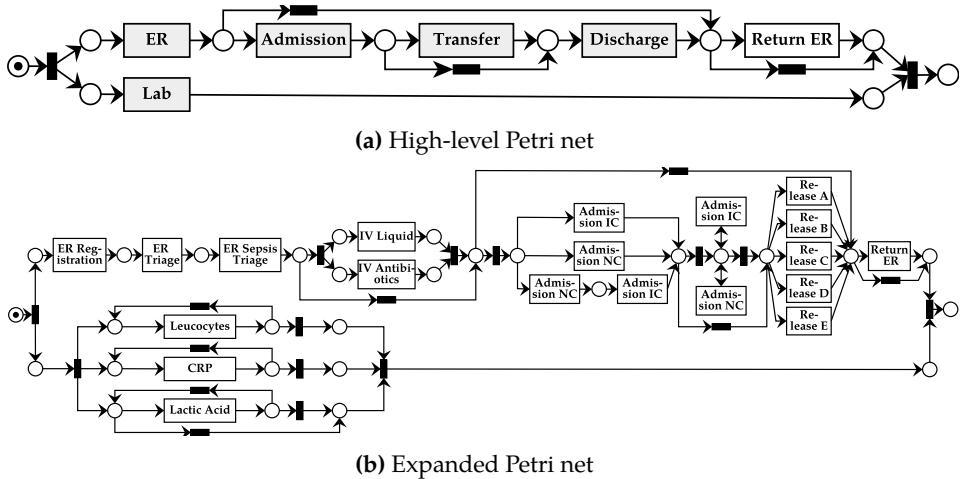


Figure 13.12: High-level and expanded Petri net discovered using IM when applying the GPD method. Gray transitions are abstracted high-level activities.

Next, we discovered the guided process model shown in Figure 13.12a on the abstracted event log using the Inductive Miner⁶⁷. This process model describes the trajectory of a patient on a high level of abstraction. Results of blood tests are obtained during the whole process (*Lab*). First, patients are in the emergency room (*ER*). Then, patients are either admitted to a hospital ward (*Admission*) or they leave the hospital. Admitted patients are possibly transferred (*Transfer*) to another hospital ward and eventually discharged (*Discharge*). Finally, patients may return to the emergency room at a later time (*Return ER*). The process model matches the high-level description of the process by stakeholders from the hospital that we obtained beforehand.

We expanded the high-level activities of the guided process model with the corresponding activity patterns as described in Section 9.2.8. Figure 13.12b shows the resulting expanded process model. We validated the quality of the discovered process model by measuring the average fitness (0.978) and the average precision (0.338) of the expanded process model with regard to the original event log. The process model fits most of the behavior seen in the event log. Thus, we can use the expanded process model to reliably answer the three initially posed process questions. Based on the average precision score, the models seems to be very imprecise. However, note that almost all of the imprecision stems from the laboratory activities *CRP*, *Leucocytes*, and *LacticAcid*. These activities may be executed in parallel to the remainder of the process. When focusing on the remaining activities and abstracting from the exact behavior of the high-level activity *Lab* the model is is

⁶⁷We used the Inductive Miner infrequent with a noise threshold of 0.2 (the default setting).

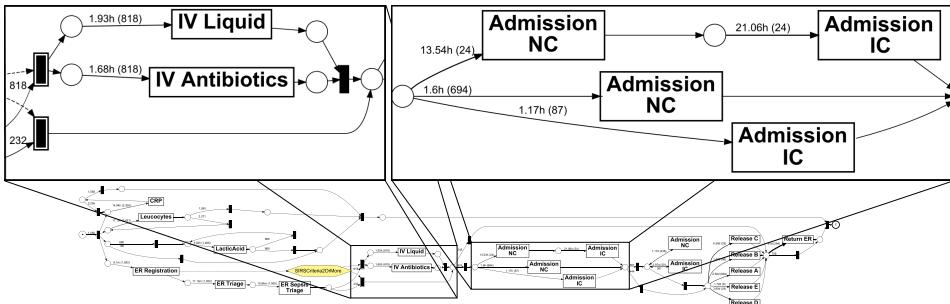


Figure 13.13: Performance information and a decision rule projected on the expanded model discovered for the sepsis event log.

more precisely captured.

Category of Admission. The first question on how patients are initially admitted to the hospital wards can be answered using the expanded process model. We projected the low-level event log on the process model using the MPE as described in Section 11.2. Figure 13.13 shows the output of the MPE. We could determine that 24 (2.9%) of the admitted patients are of the problematic category: They are first admitted to the normal care ward and, then, re-admitted to the intensive care ward. Around 694 (86.2%) of the patients are admitted to the normal care ward and 87 (10.8%) of the patients are admitted to the intensive care ward. Moreover, we used the filtering capabilities of the MPE to visualize only the trajectories of the problematic patients. This revealed that 56.5% of these patients return to the emergency room within one year (i.e., activity Return ER). Among the other patients only 27.4% return. Thus, patients of the problematic category return more often. The hospital could, e.g., monitor these patients more closely.

Infusions. We used the discovered process model to investigate whether antibiotics and liquid infusions are given to patients with a sepsis condition. There are several criteria that are checked to determine a sepsis. The event log contains the attribute *SIRSCriteria2OrMore*, which indicates whether two or more of these criteria are fulfilled. We used the MPE to retain only cases for which *SIRSCriteria2OrMore* is *true* and projected these cases on the expanded process model. This revealed that 95.3% of patients for which two or more SIRS criteria have been recorded eventually get an antibiotics infusion. However, according to the event log 15% of those patients do not receive a infusion of liquid (i.e., the alignment includes a model move for activity IV Liquid). To investigate the delay between filling in the triage form and the infusion activities, we projected the average time between activities in the entire event log on the process model. This revealed that it takes,

on average, 1.68 hours until the antibiotics are administered (Figure 13.13). When reporting both findings to the hospital, we found that data about the infusions is entered manually into the ERP system. Therefore, it is unclear whether the average time represents the real waiting time. Moreover, missing liquid infusions are, most probably, simply not registered.

Decision Rules. We also applied our decision mining techniques presented in Section 10.1 to discover decision rules for the decision points in the expanded process model. We discovered that it depends on the attribute *SIRSCriteria2OrMore* whether patients receive infusions. This can be expected as patients with more than two criteria for a sepsis should definitely receive infusions. We also discovered decision rules regarding the three different variants of admission. We found rules for the admission of patients to the normal care and intensive care. However, we did not find a good rule for the problematic category based on the attributes in the event log.

Overall, using the process model shown in Figure 13.12b we could provide meaningful answers to the process questions. For example, we found that the data quality of the *IV Antibiotics* events needs to be improved to show that the hospital acts according to their own standards. By guiding the process discovery with activity patterns, we show that it is possible to discover a model that is comprehensible to domain experts. By visualizing the patient trajectories on such a comprehensible model, it is possible to discuss the obtained diagnostics together with the process stakeholders.

13.5.3 Comparison to State-of-the-Art Methods

We applied four state-of-the-art discovery methods directly to the original low-level event log: the Heuristics Miner (HM) [WR11], the ILP Miner [ZDA15], the Inductive Miner (IM) [LFA16], and the Evolutionary Tree Miner (ETM) [Bui14]. We compared the insights that can be obtained from the process model that was discovered on the original event log with the insights gathered from the process model shown in Figure 13.12b. We used the expanded process model to compare our results to those returned by state-of-the-art methods on the same abstraction level. Thus, we ignore the added benefit that our method provides a high-level process model.

Heuristics Miner (HM). The process model discovered by the HM (Figure 13.14) is unsound, it contains unbounded behavior that prevents the process from completing.⁶⁸ Process models that are unsound are not suited for a wide range of analysis tasks [Aal16]. Therefore, we could not use the process model discovered by the HM. We still calculated the fitness (0.519) and precision measure (0.763), which

⁶⁸We used the ProM 6.7 plug-in *Mine for a Heuristics Net using Heuristics Miner*.

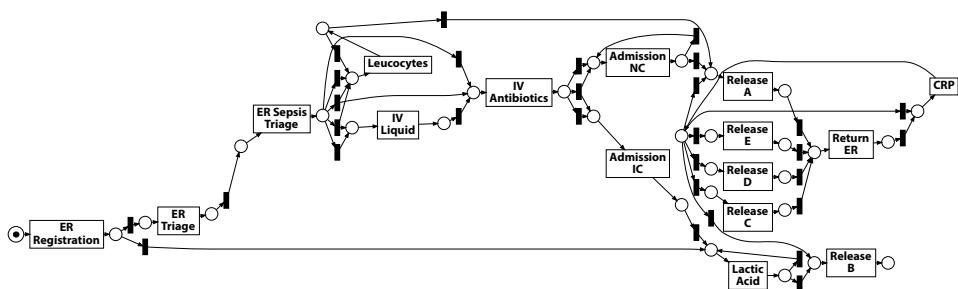


Figure 13.14: Unguided Petri net discovered using HM without applying the GPD method.

show that even when considering the subset of valid firing sequences the model does not fit the observed behavior.

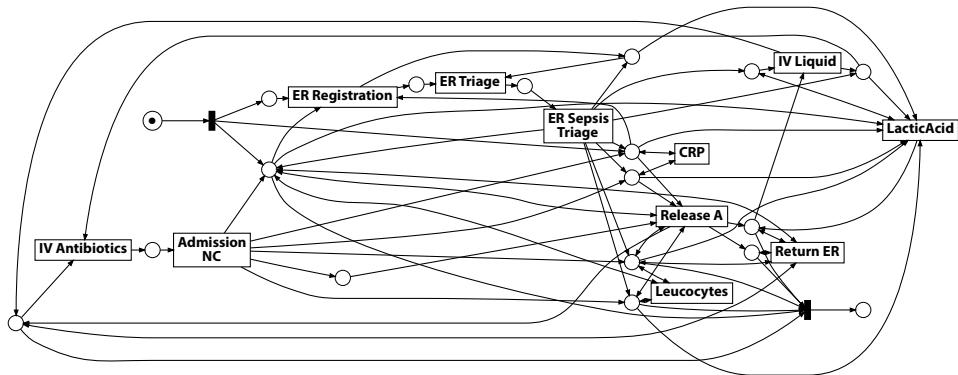


Figure 13.15: Unguided Petri net discovered using ILP Miner without applying the GPD method.

ILP Miner. The process model discovered by the ILP Miner (Figure 13.15) is precise (precision 0.804) and is fitting the event log well (fitness 0.803), but it is very complex with several non-free choice constructs⁶⁹. This made it impossible to explain it to the stakeholders (i.e., doctor and data analyst) of the hospital. Thus, the model is unsuitable in our case.

Evolutionary Tree Miner (ETM). The process model discovered by the ETM (Figure 13.16) after running for 100 generations is fitting the event log well (fitness 0.838) and is relatively precise (precision 0.743). However, it was missing the infre-

⁶⁹We used the ProM 6.7 plug-in *ILP-Based Process Discovery (Express)*.

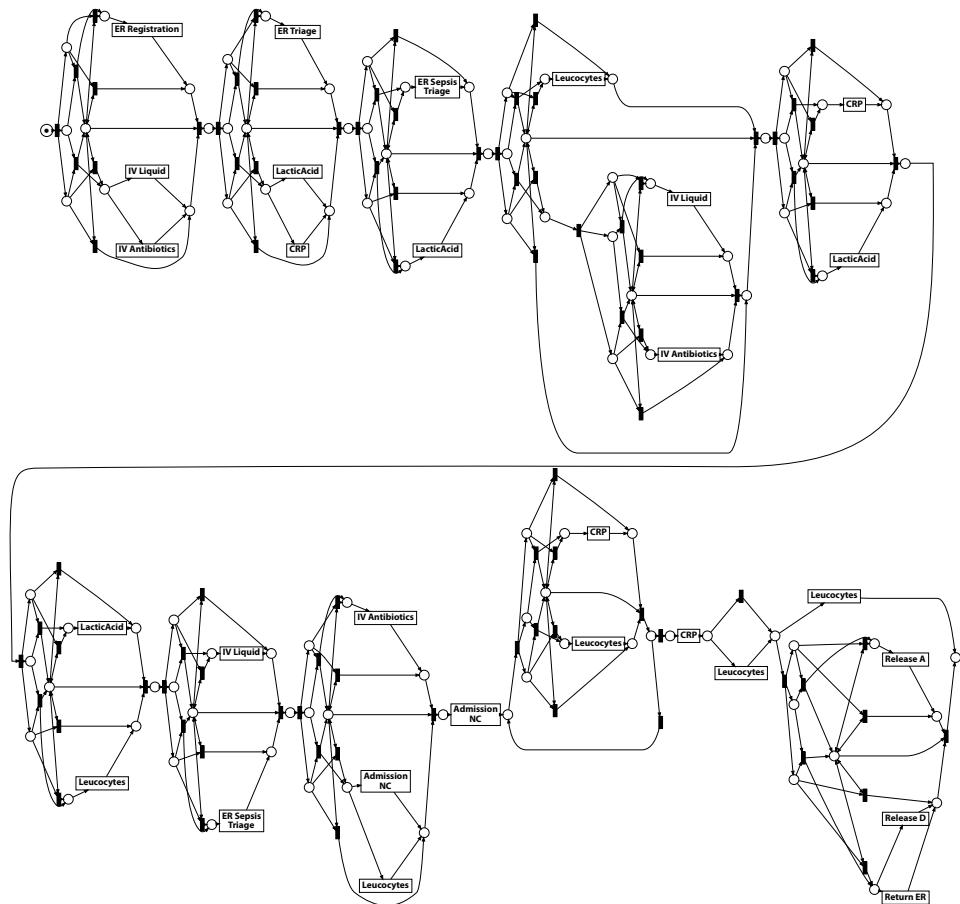


Figure 13.16: Unguided Petri net discovered using ETM without applying the GPD method.

quently occurring activities *Admission IC*, *Release B*, *Release D*, and *Release E*.⁷⁰ Since these activities are an important part of the process this model could not be used.

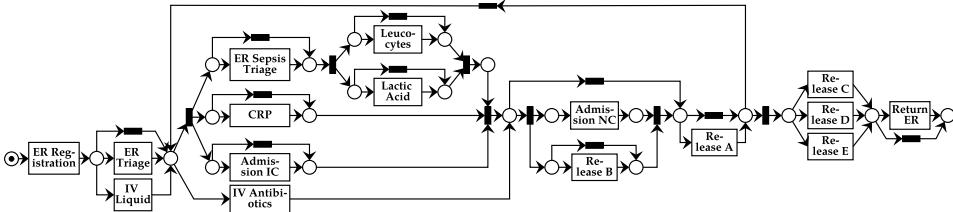


Figure 13.17: Unguided Petri net discovered using IM without applying the GPD method.

Inductive Miner (IM). Figure 13.17 shows the process model discovered by the IM.⁷¹ Among the state-of-the-art methods, we consider this the best model for our purposes. Still, note how difficult it is to use this process model to directly answer any of the process questions posed before. The model does have a comparable average fitness (0.994) and a slightly higher precision (0.470), but it fails to properly reflect the structure of the process. Semantically related activities are not grouped together since the IM does not take the organizational information and the domain knowledge on the admissions into account. For example, antibiotics and liquid infusions are placed on different decision points and the blood tests are placed within the main process flow. Moreover, it is possible to repeat most of the process after the two discharge activities *Release A* and *Release B* occurred. We know from the stakeholders that administering antibiotics is not repeated in the context of the treatment in the emergency room and this is supported by the event data. Based on the model in Figure 13.17 it is impossible to answer the first question on the problematic category of patients. Similarly, it is difficult to answer the second question on the antibiotics and liquid infusions as the process model does not contain a decision point for the infusions. The application of decision mining (i.e., the third question) requires suitable decision points to be present in the process model. The decisions modeled in Figure 13.17 are on a very low level of abstraction, i.e., on the level of skipping a single low-level activity. Therefore, we were not able to find the decision rules described in the previous paragraph. To conclude, traditional unguided process model discovery is less suited for the analysis questions we have in mind.

⁷⁰We used the ProM 6.7 plug-in *Mine Pareto* from with ETMd.

⁷¹We used the same parameter settings as when using the IM in the GPD method.

13.6 Conclusion

In this chapter, we showed that we could successfully apply *all our proposed methods* to the sepsis cases event log. We used them in the context of a case study conducted with a regional Dutch hospital.

- In Section 13.2, we analyzed a hand-made normative process model and two constraints on the time perspective with our multi-perspective conformance checking method. We used the cost function to indicate that violations on the time perspective are more likely explanations for the observed behavior than violations on the control-flow perspective. The analysis showed that the medical guideline for the timely administration of antibiotics was violated in several cases. However, we also found that the quality of the recorded timestamps was poor due to manual data entry.
- In Section 13.3, we showed that our decision mining method discovered several guard expressions. One of the guard expressions is overlapping, which highlights that the discovery of overlapping rule is relevant in practice.
- In Section 13.4, we showed that two conditional dependencies could be revealed by the proposed DHM method. These conditional dependencies, which are observed infrequently, would not have been revealed by state-of-the-art heuristic process discovery methods. Moreover, the DHM method provided decision rules regarding both dependencies that could be interpreted by the users.
- In Section 12.5, we used the GPD method to discover a process model that is meaningful for the process stakeholders by using domain knowledge on the function perspective in the form of activity patterns. We showed that we could semi-automatically discover a useful process model without requiring as much domain knowledge as necessary to create the the normative model used in Section 13.2. The discovered process model was recognizable for the stakeholders and can be used to investigate relevant questions on the process whereas state-of-the-art process discovery techniques return models that are less suited for our analysis questions.

Overall, all proposed methods could be successfully applied and provided actionable insights such as problems with the data quality of the antibiotics infusions and useful visualizations of patient trajectories.

14 Case Study: Digital Whiteboard

We also applied the methods proposed in this thesis to a real-life event log that we obtained from a *digital whiteboard system* of a large Norwegian hospital. An initial version of this case study was published in [Man+16c]. In Section 14.1, we describe the context of the case, specific process questions, and the event log. Then, we describe how we applied our GPD method in Section 14.2.

14.1 Case Description

We conducted this case study with data obtained from a digital whiteboard system of a Norwegian university hospital. The regional hospital has about 1,000 beds at several locations and is visited by about 750,000 patients per year.

Digital whiteboard systems are used to improve health care processes by raising situation awareness among nurses and to support coordination of care [Won+09]. In our case, we the digital whiteboard supports the daily work of nurses in the observation unit of the hospital. We conducted this project together with a project manager of the hospital who was responsible for the digital whiteboard system. We visited the observation ward of the hospital to observe how the system is used in the daily work practice of nurses.



Figure 14.1: Screenshot of the digital whiteboard software that is used by the Norwegian hospital with a similar structure.

A screenshot of the whiteboard software⁷² is shown in Figure 14.1. The whiteboard is used to manage information about admitted patients. Information is displayed in a tabular manner, where each row shows information about a single patient. The cells are used for various purposes, such as displaying logistical and medical information about the patient. Furthermore, a call signal system is integrated with the whiteboard. The current state of the call signal system is shown on the whiteboard. The call signal system consists of several buttons that are located in each room. These buttons are used by nurses to indicate their presence in the room or to call for assistance. Moreover, patients may also trigger an alarm directly. Generally, there are few constraints on how the whiteboard is actually used.

14.1.1 Process Questions

The main two questions that we wanted to answer with this case study were

1. How is the whiteboard system actually used by nurses in their daily work?
2. How is the integrated call signal system used by the nurses?

We were told by the project manager that there are likely to be different work practices with regard to the call signal system. Some nurses would not use the full functionality and prefer a *quick* way of working. We will elaborate on this difference between the *normal* and *quick* usage when presenting our results.

14.1.2 Event Log

We obtained data from the database of the whiteboard in the observation unit and created an event log with 8,487 *traces* and 286,000 *events* recorded between 04/2014 and 12/2015. Each trace records events for the visit of a single patient. On average, traces contain 34 events. Events are recorded for changes of single cells of the whiteboard. Such very fine grained logging leads to a low-level event log. Events in the log do not directly represent recognizable activities. In total, there are 42 distinct low-level activities in the log.

Varying work practices among nurses lead to changing events being recorded for the same high-level activity. The event log is challenging for any kind of process analytics as the semantics of results are not clear to process workers. We do not list individual names of the low-level events since we explain them when applying steps 1–5 of our GPD method in the next section.

14.2 Abstraction and Guided Process Discovery

This case study triggered the development of the event abstraction method that is used at the core of the GPD method. The pre-requisite to apply any process mining

⁷²The whiteboard system used is *Imatis Visi*: <http://www.imatis.com>

technique are events at the right level of abstraction. Therefore, we first describe how we created an abstraction model based on our domain knowledge (Section 14.2.1) and, then, report on the results of the event abstraction (Section 14.2.2).

14.2.1 Abstraction Model

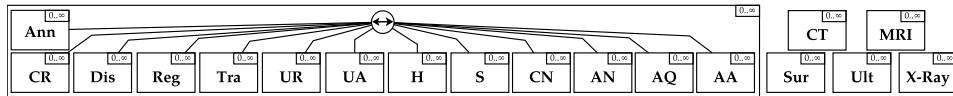


Figure 14.2: Abstraction model used in the case study. Most activities can only be interleaved (i. e., they are not concurrent) as there is only one nurse assigned to a patient.

We created an abstraction model with 18 activity patterns as shown in Figure 14.2 to apply the GPD method. The activity patterns are based on information on the whiteboard system and interviews with a domain expert from the hospital, who observed the actual work of nurses. Note that the designed activity patterns do not cover all 42 low-level activities, i. e., we considered some events recorded by low-level activities not to be relevant for our analysis.

Table 14.1: Activity patterns used for the digital whiteboard case study. The number of process activities, the number of low-level activity names shared with other patterns, and the number of recognized activity instances and the matching error are listed.

	Activity Name	Transitions (Shared)	Matches	Matching Error (ϵ)
Logistics	Announcement (Ann)	8 (6)	29	0.02
	Change Room (CR)	5 (4)	662	0.09
	Discharge (Dis)	7 (4)	8,054	0.0
	Registration (Reg)	6 (6)	9,855	0.01
	Transfer (Tra)	6 (6)	575	0.09
	Update Report (UR)	4 (0)	6,912	0.0
	Update Arrival (UA)	5 (1)	4,626	0.0
Call System	Handover (H)	1 (1)	24,228	0.0
	Shift (S)	3 (3)	405	0.04
	Call Nurse (CN)	2 (2)	12,416	0.08
	Alarm Normal (AN)	3 (3)	8,842	0.02
	Alarm Quick (AQ)	2 (2)	12,730	0.0
	Alarm Assist (AA)	5 (3)	32	0.17
Medical	CT	4 (2)	1,443	0.0
	MRI	4 (2)	124	0.0
	Surgery (Sur)	3 (3)	297	0.17
	Ultrasound (Ult)	5 (3)	1,164	0.0
	X-Ray	4 (2)	1,117	0.0

All 18 activity patterns are listed in Table 14.1 together with the number of transitions modeling low-level activities, the number of shared transitions, and the name of the modeled high-level activity. The last two columns of Table 14.1 shows

statistics about the results of our abstraction method, which we will elaborate on in Section 14.2.2. The activities can be grouped into three categories:

1. those related to patient logistics,
2. those related to the call signal system and nurse handovers, and
3. those related to ordered examinations and surgeries.

Figure 14.3 depicts three of the activity patterns that are used for patient logistics high-level activities: the registration of the patient, a transfer to another ward, and the discharge of a patient. All three activity patterns share the low-level activity *Bed Status*, which indicates that the occupancy status of the bed assigned to the patient has changed. Also the low-level activity *Transfer* is shared by all three activity patterns. Note that the system records duplicate low-level events for *Bed Status*, *Transfer*, and *Move to History*. Two wards and two digital whiteboards are involved in both the high-level activity *Transfer* and *Discharge*. The events are recorded by both systems. We use the data perspective to distinguish between the *Transfer* and the *Discharge* high-level activities: variables *Org1* and *Org2* are mapped to the same attribute in the event log that records the identifier of the whiteboard from which the event occurred. Amongst other indicators, we use this identifier to distinguish an execution of the high-level activity *Transfer* from an execution of the high-level activity *Discharge*. For the *Discharge* activity the special organizational identifier 207 is used. Figure 14.5 shows three examples of activity patterns that were created to capture medical high-level activities related to the examinations ordered for the patient. Finally, Figure 14.4 shows three examples for activity patterns related to the call signal system. In fact, we used these activity patterns as running examples (ap_a , ap_b , and ap_c) for illustrating our method in Chapter 9.

14.2.2 Event Abstraction

We applied the first four steps of the GPD method (cf., Chapter 9) to the event log and successfully obtained a smaller abstracted event log with 206,054 high-level events for 103,027 activity instances (i.e., each instance has a *start* and a *complete* event). In the context of this case study, we did not execute the expansion and validation step of the GPD method. The computation of the abstracted event log took one hour and used 6 GB of memory. We decomposed the DPN of the abstraction model into two smaller DPNs that did not share labels. The overall fitness with regard to the log was 0.91, which indicates that most of the observations could be explained. Even though 9% of the events could not be associated with any activity pattern, this is a good result for further analysis. Due to the flexibility of the information system, we can expect the event log to contain a considerable amount of noise, i. e., events unrelated to any modeled high-level activity.

The abstracted event log contains 25 high-level activities: 18 activities were obtained through abstraction and 7 further activities were already at the appropriate level of abstraction. Table 14.1 shows the resulting number of activity instances that

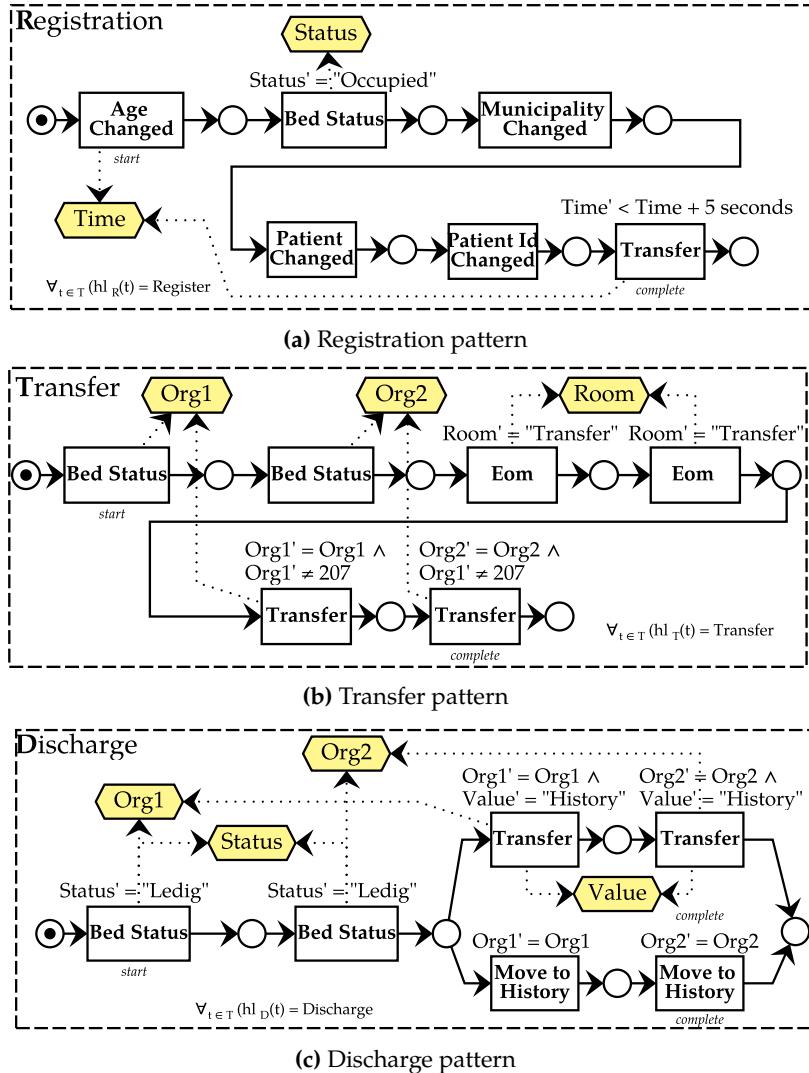


Figure 14.3: Three activity patterns that model high-level activities regarding the patient logistics: *Registration*, *Transfer*, and *Discharge*. The system records duplicate low-level events since two wards are involved in both the *Transfer* and *Discharge* activity.

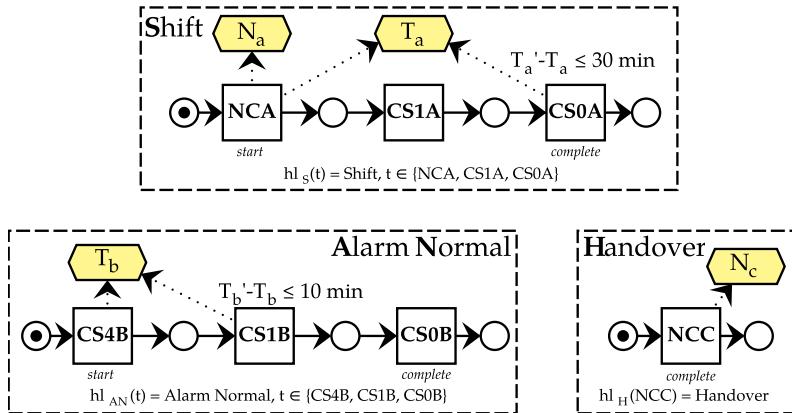


Figure 14.4: Three activity patterns that model high-level activities in the nurse call system: Shift, Alarm Normal, and Handover. We already used these patterns under the names ap_a , ap_b , and ap_c in Chapter 9 to illustrate our method.

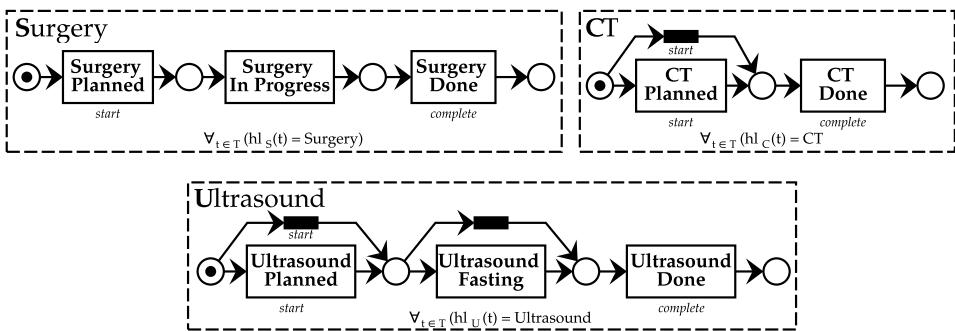


Figure 14.5: Three activity patterns for the diagnostic high-level activities: Surgery, CT, and Ultrasound. Similar patterns have been modeled for the other diagnostics-related activities.

were matched, as well as the corresponding matching error. It should be noted that the relatively high error for the activity *Surgery* stems from the fact that this activity is sometimes recorded in a different manner, i.e., one event is missing. Regarding the error for activity *Alarm Assist* we found that the *assist* button can be pressed without a prior alarm by the patient, which is different from our initial assumption.

Given the absence of a perfectly abstracted event log as ground truth, we evaluate our method by comparing the results obtained using three process analytics techniques *with* and *without* the abstraction. Using the abstracted event log, we obtained several insights into work practices of nurses in clinical processes. A domain expert from the hospital stated that the analysis: “[...] gives insight beyond the usual reports and analysis that we have access to. It gives a fresh and “new” perspective on how we understand the processes involved in running a ward or department.” By contrast, we show that using the low-level event log directly does not lead to any insights for stakeholders, because the semantics of low-level events are unclear.

We explored the high-level event log by discovering process models using the *Inductive Miner* with its life-cycle extension [LFA16] (Section 14.2.3), checked the conformance of the discovered model (Section 14.2.4), and used dotted charts [SA07] to answer a specific process question about the nurse call system (Section 14.2.5).

14.2.3 Discovery of the Inductive Miner

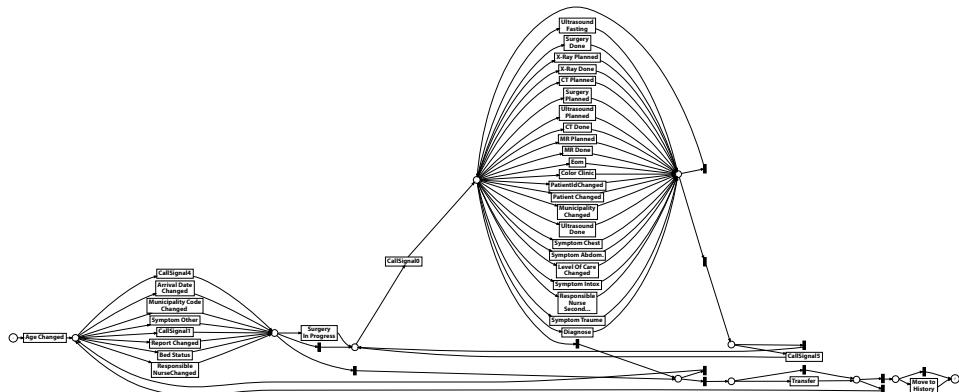


Figure 14.6: Petri net discovered for the low-level event log.

We analyzed two parts of the whiteboard system by discovering process models by using the *Inductive Miner* [LFA16] with its life-cycle extension.⁷³ We used only those events from the original event log that are used in the respective activity patterns, i.e., not all 42 low-level activities are present in the filtered event log. This

⁷³We used the *Inductive Visual Miner* plug-in of ProM 6.7 to discover models.

indicates what results could be obtained by only filtering the original log based on some knowledge about the low-level events. For both event logs we used the same noise filtering settings. Since the process is very flexible we focus on the main behavior and set the noise filtering parameter to 0.5 and use a frequent traces pre-filter to consider only 50% of the most occurring traces.

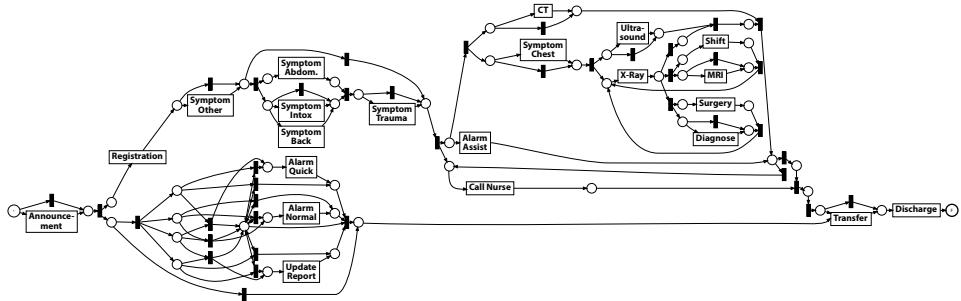


Figure 14.7: Petri net discovered for the high-level event log.

Figure 14.6 shows the Petri net discovered by the Inductive Miner for the low-level event log and Figure 14.7 shows the Petri net discovered for the high-level event log. Note that some activities in the input event log were already at the desired level of abstraction. Therefore, the model in Figure 14.7 includes more than 18 activities. The overall fitness of the high-level model with regard to the high-level log is 0.66 and the overall precision score is 0.60. In comparison, the process model discovered on the filtered low-level log fits better (fitness 0.89) since it allows for almost any behavior but has a poor precision score of 0.25.

Apart from the measures, the low-level process model gives little insights into the usage of the whiteboard system. Almost all of the activities may be repeated in any order. The high-level model in Figure 14.7, instead, contains recognizable activities that can be used to investigate the usage of the call signal system further. For example, it is visible that some patients are announced before they are registered and that the process generally starts with the registration of the symptoms of the patient. In parallel to this, the call signal system may be used and the report column might be updated. The complicated control-flow structure with many invisible routing transitions in this branch is created by the Inductive Miner because it discovered an OR-split, i.e., any combination of the three activities *Alarm Quick*, *Alarm Normal*, and *Update Report* may be executed. Another structure that was discovered is that for multiple patients the planning and execution of *surgeries* and updating the *diagnose* occur together in parallel. Finally, patients are discharged and for some patients a transfer is prepared. By using activities on the same abstraction level, the process model in Figure 14.7 offers a better insight into the process. Moreover, it allows to discuss the observations with process workers.

We also used the MPE to project frequencies based on an optimal alignment

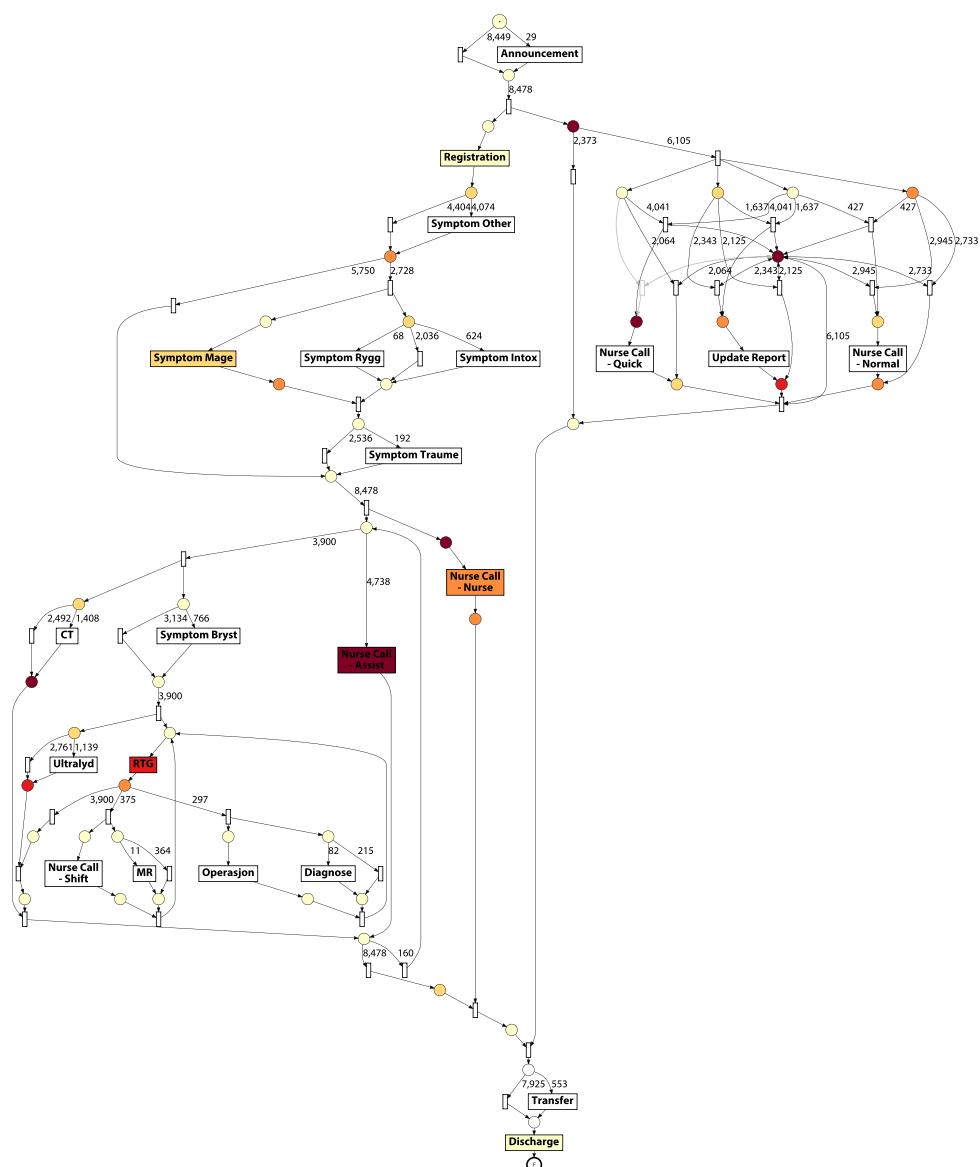


Figure 14.8: Output of the MPE fitness model for the Petri net discovered from the high-level event log.

(here only the control-flow is aligned) onto the model. Figure 14.8 shows the output model. Transitions are colored according to their local fitness score (cf., Section 11.2) and their frequency is written on the incoming edges. This revealed that the local fitness measure of the *X-Ray* and *Alarm Assist* activities in the discovered high-level model is very poor. Thus, events are missing for executions of these activities.

14.2.4 Conformance Checking of the Discovered Model

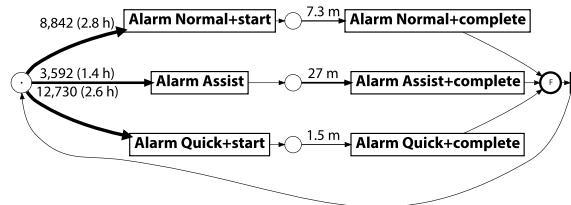


Figure 14.9: Average time between events projected on a process model modeling the usage of the call signal system.

Finally, analyzed differences between the work practice of nurses when responding to patient-initiated call signals using the MPE (Section 11.2). Figure 14.9 shows the output of the MPE performance mode for a small process model with three variants. Indeed, there are differences between activities *Alarm Normal* (AN) and *Alarm Quick* (AQ). For activity AN the nurse first indicates her presence in the room by using a button on the call signal system, after which she attends the patient. However, within activity AQ nurses do not use this functionality. The average service time for activity AN (7.3 min) is longer than for activity AQ (1.5 min). This might be because nurses do not use the full functionality of the call signal system for minor tasks, which may be important for the hospital to investigate further.

We were also interested in investigating whether some nurses use the *Alarm Quick* activity more often than the standard way of handling the call signal system. Figure 14.10 shows a comparison between the frequency with which nurses use *Alarm Quick* (blue bars) and *Alarm Normal* (red bars) that is created by using the MPE chart view. It seems that there are some nurses that do not always stick to the standard procedure more often than others. This has been already suspected by the domain expert of the hospital. Note that we anonymized the identifiers before conducting the case study to protect the privacy of the employees.

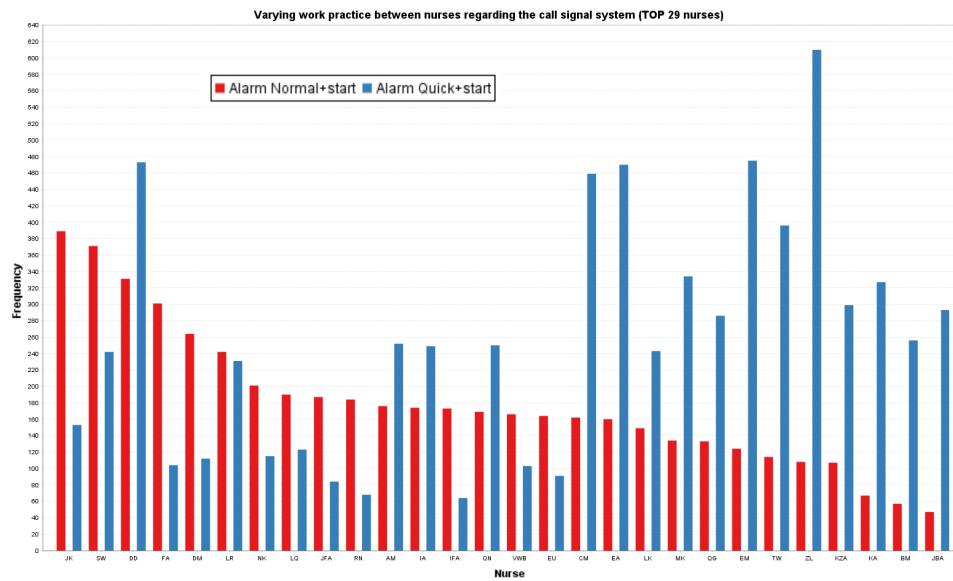
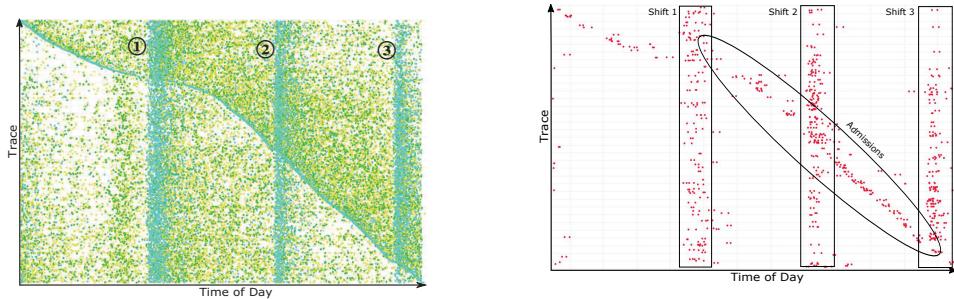


Figure 14.10: Chart view of the MPE reveals that some nurses use the *quick variant* (blue bars) of responding to an alarm instead of the desired variant (red bars) more often than others. The identifiers have been anonymized beforehand.



(a) Dotted chart of the following events in the low-level event log: *NurseCall* (NC) in blue, *CallSignal1* (CS1) in green, and *CallSignal0* (CS0) in yellow.

(b) Dotted chart of the high-level *Shift* events. Each of the vertical lines (i.e., groups of events) corresponds to a change of shifts. The events on the diagonal are caused by admissions.

Figure 14.11: Dotted charts of events related to the activity *Shift*. Traces are shown on the y-axis and sorted by the time of day of the first event in a trace.

14.2.5 Dotted Chart Analysis of the High-level Log

Figure 14.11 shows two dotted charts created with ProM.⁷⁴ Figure 14.11a was created using the original low-level event log. It shows the distribution of events NC, CS1 and CS0 over the course of a day. As expected, the NC event (i. e., the responsible nurse changed) mostly occurs when a patient is admitted (i. e., on the blue diagonal) and during one of the three shift changes (i. e., the three blue vertical lines numbered ①–③). Still, the responsible nurse also changes between those well-defined times. Yet, from Figure 14.11a it is not evident whether nurses use the call signal system when visiting a patient after their shift started. Looking at Figure 14.11b, which shows only the high-level events *Shift* from the abstracted event log, it is clearly visible that our assumption was correct. Activity pattern *Shift* captures a meaningful high-level activity. Figure 14.11b shows that nurses did use the call signal system to indicate their presence in the room of the patient after taking responsibility for a patient. In contrast to the dotted chart in Figure 14.11a, event *Shift* only occurs after admissions (dots on the main diagonal) and after shift changes (the three vertical lines named Shift 1, Shift 2, and Shift 3). Still, by comparing the number of activity instances in Table 14.1 it is clear that activity *Shift* (405 times) happens rarely in comparison to activity *Handover* (24,228 times). Two likely reasons for this are that nurses do either not attend the patient after a shift change, or that they do not use the system to indicate their presence. This is a valuable insight on how the whiteboard system is used in practice. This insight was only obtained using the abstraction method and could not be obtained by conventional approaches.

14.3 Conclusion

We applied the GPD method to the event log obtained from the digital whiteboard system. We did not apply the remaining methods since the event log is missing attributes that could be used to discover relevant rules. Moreover, we did not obtain a data-aware normative model that would be suitable to test our multi-perspective conformance checking methods. However, we showed that it was possible to obtain a high-level event log based on activity patterns and alignments. Using this log, we could conduct a preliminary analysis of the work practices of nurses with regard to the digital whiteboard. Further refinement of the collected data (e. g., with suitable data attributes) and the formulation of more specific process questions would be needed to leverage the remaining methods proposed in this thesis.

⁷⁴We used the ProM 6.7 plug-in *Log Projection*.

15 Case Study: Hospital Billing

In our forth case study, we applied the methods proposed in this thesis to a real-life event log that we obtained from the financial modules of the ERP system of a regional hospital in The Netherlands. Parts of this case study were published in [Man+17].

This chapter is structured as follows. First, we describe the context of the case, specific process questions, the extracted event log, and a handcrafted normative process model in Section 15.1. Since the designed normative process model does not include any multi-perspective constraints, we started by applying our proposed decision mining method in Section 15.2. Then, we check the conformance of the enhanced process model with decision rules in Section 15.3 in terms of multi-perspective precision and fitness. We also describe the application of the DHM method (Section 15.4) to the event log.

15.1 Case Description



Figure 15.1: The *desired path* through the billing process runs through five states.

We conducted this case study in the context of the billing process for medical services in a regional hospital in The Netherlands. In the process under observation, several medical services (e.g., medical diagnostics, hospitalization, treatment, surgeries, etc.) are collected in a billing package⁷⁵ and billed together after a certain amount of time has passed or the treatment ends.

Figure 15.1 depicts the *desired path* of the billing process in which the billing package runs through five states. Newly created billing packages (i.e., instances of a combination of a diagnose and the treatment) start in the *in progress* state. These packages need to be *closed* and *released* before an invoice can be sent. Often, a separate *declaration code* needs to be obtained before the package is *billable*. This code can be used to send the package (often together with others in batches) to the

⁷⁵In the Dutch healthcare system billing packages are referred to as diagnose-behandelcombinatie (DBC), i.e., they are a combination of a specific diagnose and the corresponding treatment. For simplicity, we refer to it as billing package in the remainder of this thesis.

responsible healthcare insurance in the form of an invoice. Normally, this would be the end of the process that we analyze and the billing package would be marked as *billed* in the system.

15.1.1 Process Questions

At a first glance, the process seems to be straightforward as depicted in Figure 15.1. However, in some cases the process is more complex. The rules and regulations around the billing of medical services have changed every year. There exists a large number of special situations and mistakes are made when determining what should be the correct billing package and declaration code. Thus, in some cases the billing package is *reopened*, *rejected*, or *canceled* and further changes are needed. The goal of our case study was to visualize and analyze this billing process with our proposed methods. Specifically, we aimed to answer the questions: "Why are some billing packages reopened multiple times and their invoicing is delayed?" In the next two sections, we describe the event log extracted from the ERP system of the hospital and the manually created process model, which describes the expected variants of how the billing package is handled.

15.1.2 Event Log

We extracted an event log from the financial part of the ERP system⁷⁶ of the hospital. The event log originates from the tables of the ERP system that store information about the status and processing of billing packages. We took a random sample of 100,000 traces with events spanning a time frame of three years and anonymized the activity names and attributes. Therefore, information on the actual medical services used and the code used for declaration was removed. The resulting event log contains events for 17 activities as listed in Table 15.1 and 16 attributes as listed in Table 15.2. The anonymized event log was made available for further process mining research purposes as part of the collection of real-life event logs of the IEEE Task Force on Process Mining [Man17].⁷⁷

15.1.3 Normative Model

Figure 15.2 shows a manually created process model of the billing process that we used to visualize and analyze the handling of the billing packages by the financial department of the hospital. It is a normative model of the process in the sense of depicting the expected variants of process behavior as communicated by the stakeholders from the hospital's financial department. However, since we are investigating the internal management of the billing package, the model in

⁷⁶The employed ERP system is based on SAP and is customized for the hospital.

⁷⁷The hospital billing event log can be obtained from: <https://doi.org/10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfcb741>

Table 15.1: Activities recorded in the hospital billing event log.

Activity	Frequency	Description
NEW	101,289	A new billing package is created.
FIN	74,738	The billing package is closed, i. e., it may not be changed anymore.
RELEASE	70,926	The billing package is released to be sent to the insurance company.
CODE OK	68,006	A declaration code was successfully obtained.
BILLED	67,448	The billing package has been billed, i. e., the invoice is sent out.
CHANGE DIAGN	45,451	The diagnosis that the billing package is based on was changed.
DELETE	8,225	The billing package was deleted.
REOPEN	4,669	The billing package was reopened, i. e., additional medical services may be added or existing services removed.
CODE NOK	3,620	The declaration code was obtained with an error message.
STORNO	2,973	The billing package was canceled.
REJECT	2,016	The invoice sent to the insurance company was rejected.
SET STATUS	705	The status (i. e., new, closed, etc.) was manually changed.
EMPTY	449	The status (i. e., new, closed, etc.) was manually changed.
MANUAL	372	The billing package was manually changed from a non-standard system.
JOIN-PAT	358	Two billing packages are joined together since they refer to the same patient.
CODE ERROR	75	The declaration code could not be obtained.
CHANGE END	38	The projected end date of the billing package was changed.

Table 15.2: Attributes recorded in the hospital billing event log.

Attribute	Domain	Description
actOrange	boolean	A flag that is used in connection with services that may not be covered by the standard health insurance.
actRed	boolean	A flag that is used in connection with services that may not be covered by the standard health insurance.
blocked	boolean	A flag that is used when the billing may not proceed (i. e., is blocked).
caseType	literal	A code for the type of the billing package, which may influence its handling.
closeCode	literal	There may be several reasons to close a billing package, this attribute stores the code used.
diagnosis	literal	A code for the diagnosis used in the billing package.
flagA	literal	An anonymized flag.
flagB	literal	An anonymized flag.
flagC	literal	An anonymized flag.
flagD	literal	An anonymized flag.
isCancelled	boolean	A flag that indicates whether the billing package was eventually cancelled.
isClosed	boolean	A flag that indicates whether the billing package was eventually closed.
msgCode	literal	The code returned by activity CODE NOK.
msgCount	discrete	The number of messages returned by activity CODE NOK.
msgType	literal	The type of messages returned by activity CODE NOK.
speciality	literal	A code for the medical speciality involved.
state	literal	Stores the current state of the billing package.
version	literal	A code for the version of the rules used.

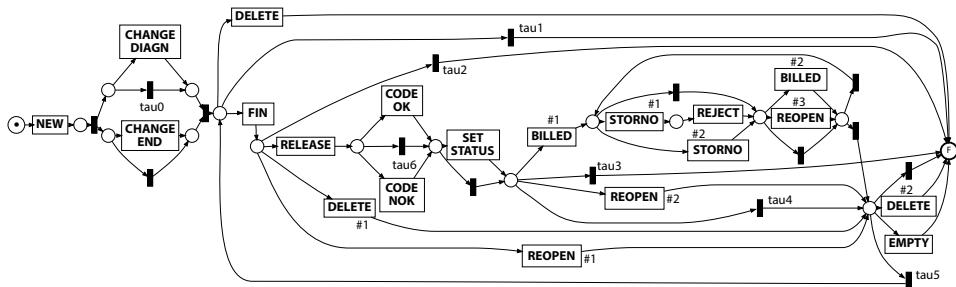


Figure 15.2: Manually created process model for the billing process that we used for visualization purposes and decision mining.

Figure 15.2 does not encode rules and regulations for the billing of medical services. The process starts with creating the billing package (*NEW*). Sometimes, the registered diagnosis needs to be adjusted after the billing packages has been created (*CHANGE DIAGN*). Moreover, for some billing packages the initially determined end date⁷⁸ is changed. Thereafter, the process continues with its main part. Some of the billing packages are directly deleted (*DELETE*) at this stage of the process and for some the processing ends (*tau1*). In the remaining cases the package is closed after some time (*FIN*). There are several options to continue:

1. the processing ends (*tau2*),
2. the package gets deleted,
3. the package gets opened again (*REOPEN*), or
4. the package gets released (*RELEASE*).

In case the package was released (option 4), the declaration code may be obtained successfully or with problems (*CODE OK* or *CODE NOK*). Afterwards, for a small number of cases the state is manually adjusted (*SET STATUS*). Again, there are several options to continue:

1. the processing ends (*tau3*),
2. the package is not directly billed (*tau4*),
3. the package is sent to the insurance company (*BILLED*), or
4. the package is reopened (*REOPEN*).

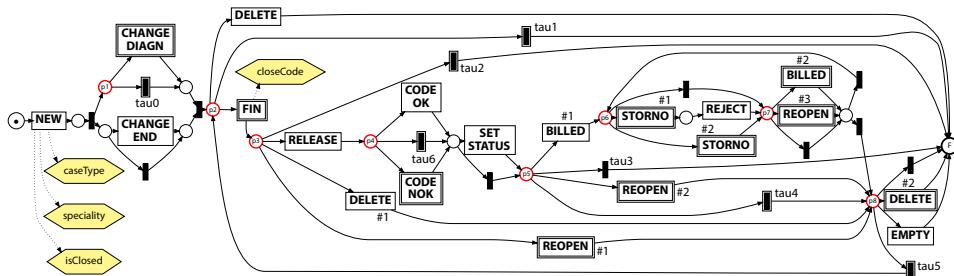
For those packages that were billed (option (3)) we model several exception variants that may occur. Sometimes invoices get rejected or need to be canceled for other reasons. Activities *STORNO* and *REJECT* are executed in these cases. Rework is needed for those billing packages, which may get reopened or directly billed again.⁷⁹ Finally, for some packages (most of them were reopened) the processing

⁷⁸Each billing package covers all medical services provided in a certain time period.

⁷⁹In these cases the rework has been done outside of the system under observation.

starts over (τ_{u6}), some packages are declared empty (*EMPTY*) or deleted (*DELETE*), and for the remaining packages the process ends.

15.2 Discovery of the Data Perspective



(a) DPN of the hospital billing process enhanced with guards and write operations.

Transition	Guard expression
CHANGE DIAGN	
tau0	caseType = B
	caseType ≠ B
tau1	speciality = K
	speciality ≠ K
FIN	
	(caseType ≠ F ∧ caseType ≠ C) ∨ (caseType ≠ F ∧ caseType ≠ C ∧ closeCode = A)
tau2	closeCode = H
	closeCode ≠ H
REOPEN #1	
tau6	caseType = F ∨ (caseType ≠ F ∧ caseType ≠ C ∧ closeCode ≠ A)
CODE NOK	(caseType ≠ F ∧ caseType = C) ∨ (caseType ≠ F ∧ caseType ≠ C ∧ closeCode = A)
REOPEN #2	caseType = B ∨ (caseType ≠ B ∧ closeCode ≠ A)
tau4	caseType ≠ B ∧ closeCode = A
STORNO #1	(isClosed = true ∧ closeCode ≠ A) ∨ isClosed ≠ true
STORNO #2	isClosed = true ∧ closeCode = A
BILLED	isClosed = true
REOPEN #3	isClosed ≠ true
DELETE #2	isClosed ≠ true
tau5	isClosed = true

(b) Guards discovered by the overlapping decision mining method.

Figure 15.3: Decision rules discovered for the hospital billing process using the overlapping decision mining method. We numbered the transitions referring to the same activity to uniquely identify the transitions that were assigned a guard expression.

We applied our overlapping decision mining technique to the hospital billing event log. Figure 15.3 shows the resulting DPN in which we discovered guard expressions for 16 transitions at eight decision points (labeled p1 ... p8) of the process.⁸⁰ The discovered decision rules are based on four attributes of the process: caseType,

⁸⁰We used binary rules, set the minimum instances parameter to 0.15, and excluded the following attributes *state*, *org:resource*, *version*, *msgCode*, *msgCount*, and *msgType* to avoid overly complex rules.

speciality, closeCode, and isClosed. Some of the discovered decision rules are overlapping. Our method first discovered the guard expression true, i. e., the transition is unconditionally activated, based on a majority vote and, then, discovered a decision rule among the misclassified instances. For example, transition CODE OK is unconditionally enabled and, hence, the guard expression overlaps with the rules discovered for transitions *tau6* and *CODE NOK*. We briefly discuss the discovered rules for each of the decision points.

Place p1. Only for the cases of type B, the diagnosis is changed after creation of the billing package. Indeed, this is expected for these cases since they are created for patients without a prior treatment.

Place p2. Cases of the speciality K are not processed further. When confronting the stakeholders of the hospital with this rules, we found that these cases are not actually billed by the hospital and, thus, get assigned a *dummy* speciality.

Place p3. Cases with the close code H are not processed further. We found that this code means that the billing package is empty, i. e., no billable services were carried out.

Place p4. For cases of case type F no declaration code needs to be obtained. This could be explained, since this code is associated with intensive-care activities for which the code is, often, not required. Moreover, the code seems to be NOK for a close code of A or a case of type C. Cases of type C represent services that billed separately outside of the normal billing package process. We could not find an explanation regarding the close code A.

Place p5. Cases of type B or cases with a close code different from A are apparently reopened more often than others. We could not find an explanation for this rule.

Place p6. For cases that are not yet closed (i. e., not yet successfully finished upon taking the random sample) the variant *STORNO* and *REJECT* is observed more often. Also, the decision depends, again, on close code A. Unfortunately, the interpretation of this rule is unclear.

Place p7. Cases which have been reopened at this decision point are not yet closed in contrast to cases which are billed at this decision point.

Place p8. Deleted cases are not in the close status as to be expected.

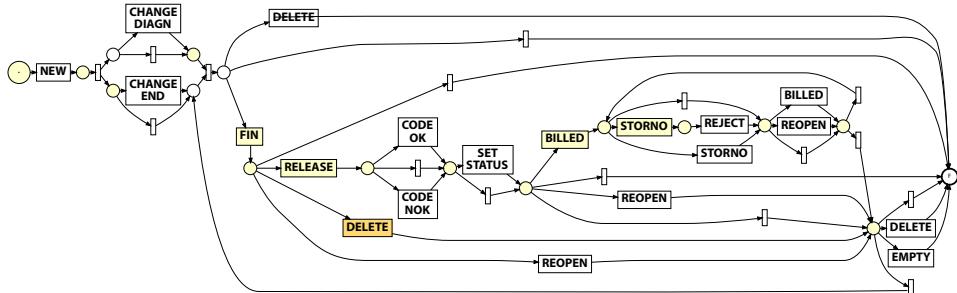
When using standard decision mining techniques such as the one presented in [LA13b], some of the insightful decision rules were not discovered. Next, we check how the precision and fitness of the process model changed when adding the discovered decision rules.

15.3 Conformance Checking

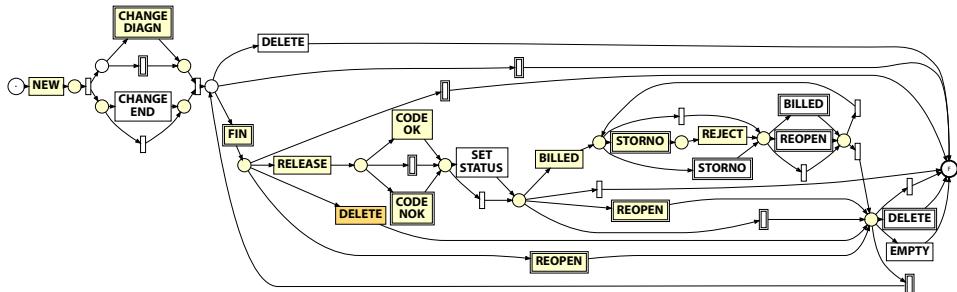
We used our multi-perspective conformance checking techniques to assess the original model (Figure 15.2) and the multi-perspective model (Figure 15.3) in terms fitness and precision score. The computation took 10 minutes. On average the fitness of the model without data rules was 0.991 and the fitness of the model with data rules was 0.987. As to be expected the discovered guard expressions were not fulfilled by the observed data in all the cases. Still, the discovered rules are representative for the majority of the cases. We also computed the multi-perspective activity-precision score (Section 6.2.3) to investigate how much precision was added by enhancing the process model with decision rules.⁸¹ The precision of the original model without data rules was 0.594, which was increased by the added decision rules in the enhanced model by 0.106 to a score of 0.700.

Figures 15.4 and 15.5 show the influence of the added decision rules by comparing the local fitness and precision scores for both models. On the one hand, the fitness score decreases for several of the output transitions of decision points that are assigned decision rules. Also, the local fitness of the transition *NEW* decreased since three of the variables that are used by the decision rules are written by this transition. In total, there are 3,898 incorrect write operations diagnosed for transition *NEW* compared to 296,102 correct write operations namely 1.2%. On the other hand, the local precision score increases for the places p_1, \dots, p_8 since fewer transitions are activated for the observed behavior, i. e., according to the control-flow and the data perspective. In conclusion, the discovered decision rules improved the precision of the process model without sacrificing too much fitness. The multi-perspective process model provides a better balance between fitness and precision.

⁸¹We only considered the attributes which were used in decision rules as sources of imprecision, i. e., $V_{PR} = \{\text{caseType}, \text{speciality}, \text{closeCode}, \text{isClosed}\}$ in both cases.

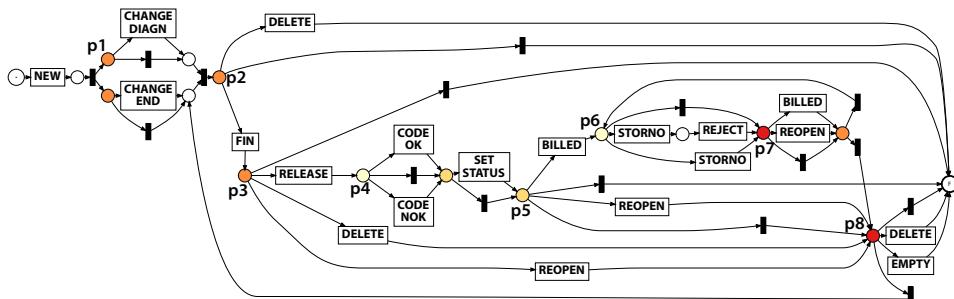


(a) Fitness measure projected on the normative model without decision rules (cf., Figure 15.2).

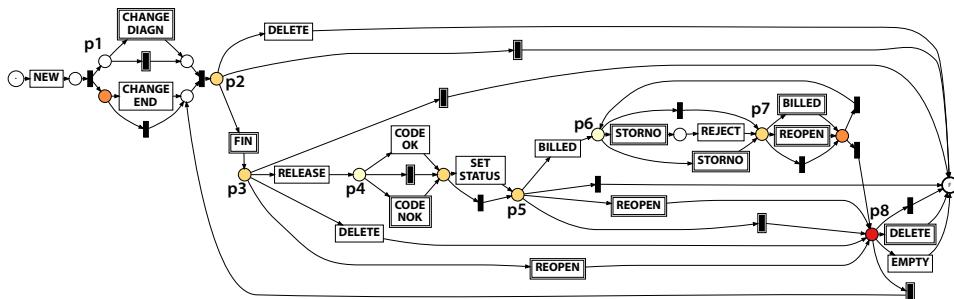


(b) Fitness measure projected on the enhanced model with partly overlapping decision rules (cf., the guard in Figure 15.3 for double-bordered transitions).

Figure 15.4: Fitness measure projected on both the hospital billing process models without and with decision rules. The local fitness score decreases for several transitions since some observed attribute assignments do not fulfill the discovered guards, e.g., transitions NEW and CHANGE DIAGN.



(a) Precision measure projected on the normative model without decision rules (cf., Figure 15.2).



(b) Precision measure projected on the enhanced model with partly overlapping decision rules (cf., the guard in Figure 15.3 for double-bordered transitions).

Figure 15.5: Precision measure projected on both the hospital billing process models without and with decision rules. The local precision score increases for several places (p_1, \dots, p_8) fewer transitions are activated for the observed attribute values.

15.4 Data-aware Process Discovery

We also applied the data-aware heuristic discovery method that was introduced in Chapter 8 to the hospital billing event log. Here, we do not use the initial normative model but try to discover a process model from the event log.

15.4.1 Configuration Settings

We experimentally determined the following parameter settings:

- the *accepted-task-connected heuristic*, as some activities might not be relevant;
- the observation threshold $\theta_{obs} = 0.04$ (i. e., relations that appear in more than 4% of the log trace), because we want to capture the main flow of the process;
- the dependency threshold $\theta_{dep} = 0.9$ to discover only strong dependencies;
- the binding threshold of $\theta_{bin} = 0.001$ to exclude very infrequent bindings.

We used 13 of the attributes⁸² to discover dependency conditions and guarded bindings. We used C4.5 with 10 times 10-fold cross validation and only accepted classifications with a conditional dependency measure of $\theta_{con} \geq 0.6$.

15.4.2 Discovery Results

Figure 15.6 shows the DC-Net discovered by the iDHM in about 3 seconds for the hospital billing event log. The discovered model fits 97% of the observed behavior.

Conditional Dependencies

Compared to the model returned by the standard Heuristics Miner, our method revealed six additional dependencies. We numbered these relations in Figure 15.6 and list the conditional dependency measure, the frequencies of occurrence according to the binding heuristic, as well as quality-scores (i. e., Cohen's kappa) and used attributes of the obtained dependency conditions for each relation in Table 15.3. We discussed the discovered conditional dependencies with a domain expert from the hospital who works in this process.

Added dependency ①. The relation from FIN to END is based on a special *closeCode* attribute that is used when nothing can be billed and, hence, the process ends.

Added dependency ②. The relation between RELEASE and CODE NOK occurs mostly for two specific *caseType* values. According to the domain expert both case types correspond to exceptional cases: one is used for intensive care and the other for cases for which the code cannot be obtained (CODE NOK).

⁸²Specifically, we included the following attributes: *caseType*, *closeCode*, *blocked*, *flagA*, *flagB*, *flagC*, *msgCode*, *msgType*, *speciality*, *isClosed*, and *version*.

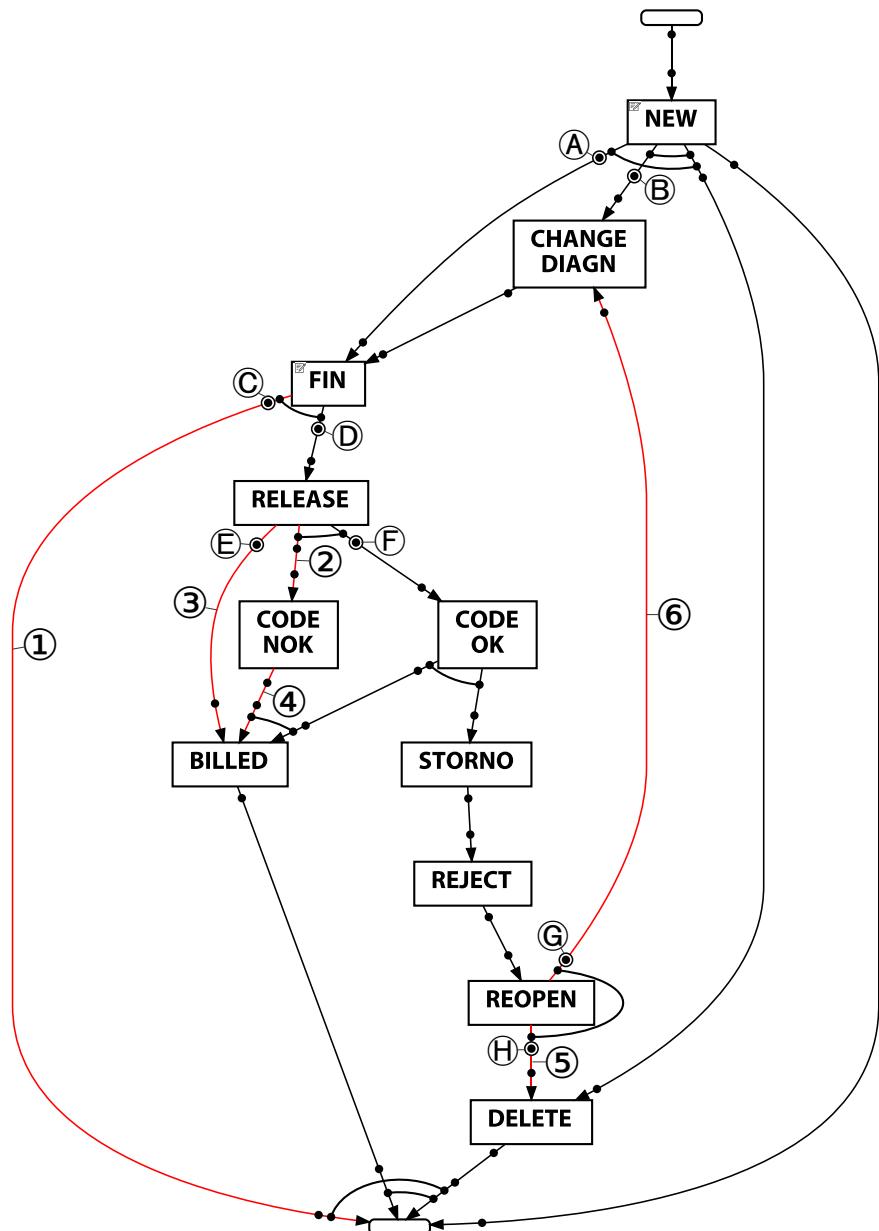


Figure 15.6: DC-Net discovered for the hospital billing event log. The numbered red edges (①–⑦) were added by the DDM. The double-bordered bindings (Ⓐ–Ⓗ) are assigned the guards listed in Table 15.4.

Table 15.3: Dependency conditions discovered for the hospital billing log.

Nr	Source	Target	Frequency	Quality	Dependency	Attributes
①	FIN	END	3,619	0.98	1	closeCode
②	RELEASE	CODE NOK	1,674	0.62	0.99	caseType
③	RELEASE	BILLED	468	0.93	0.98	caseType
④	CODE NOK	BILLED	1,481	0.84	0.99	caseType, specialty
⑤	REOPEN	DELETE	1,128	0.83	0.81	isClosed
⑥	REOPEN	CHANGE DIAGN	212	0.97	0.99	isClosed

Added dependency ③. The relation from RELEASE TO BILLED is, again, related to a specific *caseType*. This case type F is used for intensive-care activities as well and, often, does not require a code to be obtained.

Added dependency ④. The relation from CODE NOK to BILLED is also related to the *caseType*. Moreover, also the medical *specialty* is used by the classifier learned. This could not be explained by the domain expert.

Added dependencies ⑤ and ⑥. Both relation from REOPEN to DELETE and from REOPEN to CHANGE DIAGN are conditional to the attribute *isClosed*, which indicates whether the invoice is closed or not. Clearly, deleted cases should not be in the closed status, whereas reopened cases with a change in diagnosis can be eventually closed in the future.

Overall, the process model discovered by the DHM provides a balanced view on the interesting infrequent paths of the billing process together with the more frequent, regular behavior. Moreover, additional insights are provided by revealing the conditions with which infrequent paths occur.

Guarded Bindings

In addition to the conditional dependencies, we also discovered several guarded bindings. Table 15.4 lists the discovered guard expressions that are highlighted using the letters ⑧–⑩ in Figure 15.6.

The discovered guards are based on the attributes *caseType*, *closeCode*, and *isClosed* and similar to the overlapping decision rules discovered in Section 15.2. However, the setting is now different. We did not use a normative model and used different parameter settings.

Quality of the Discovered DC-Net

We computed the fitness and precision score of the discovered DC-Net⁸³. The fitness score is 0.967 and the precision score is 0.808. Thus, the DC-Net is more

⁸³As described before in Section 12.4.2, we use a DPN translation of the DC-Net for this purpose.

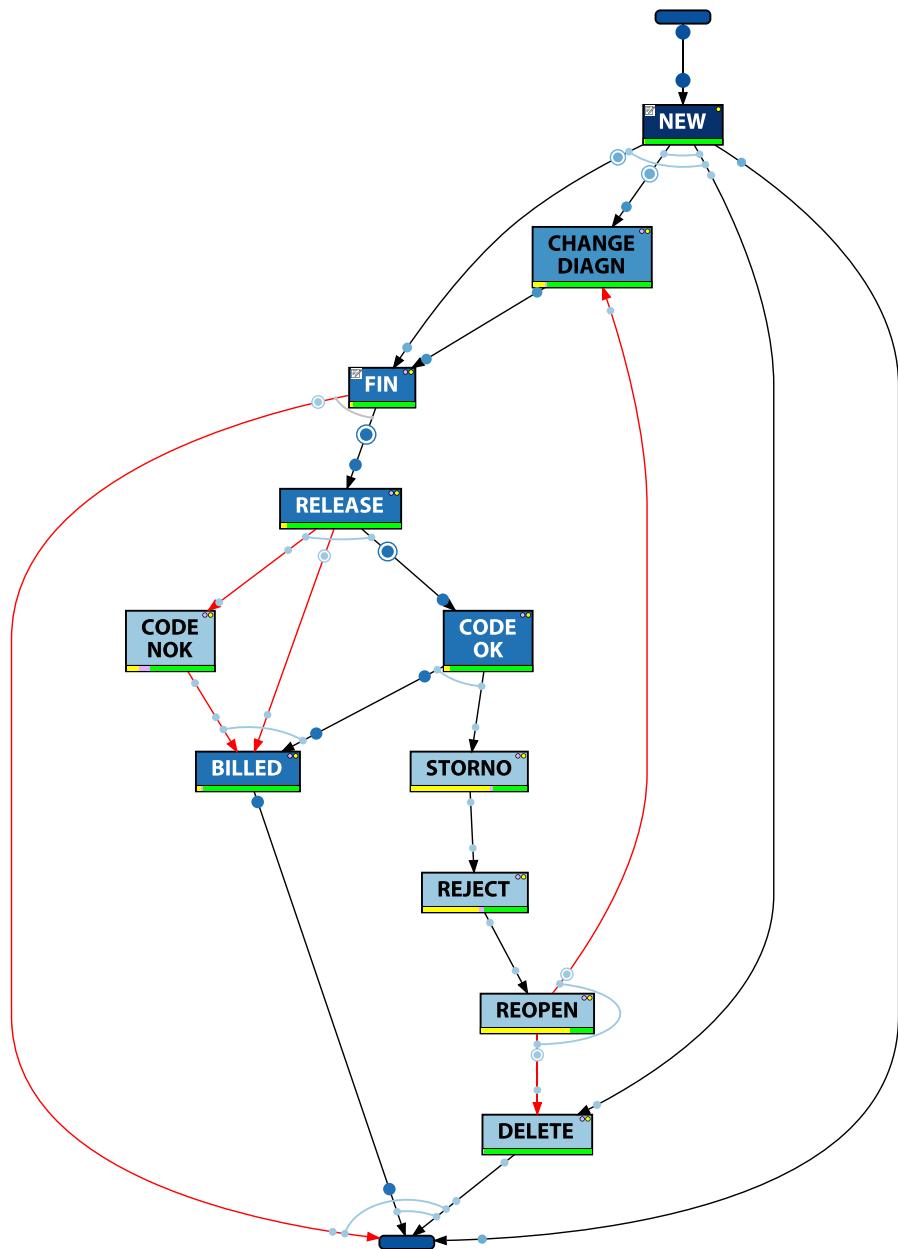


Figure 15.7: Conformance information projected on the discovered DC-Net of the billing process based on a multi-perspective alignment. Some local conformance issues are visible. Many events referring to the activities on the path from STORNO to REOPEN are diagnosed as *log moves*.

Table 15.4: Decision rules for the billing process DC-Net.

Nr	Source	Target	Guard Expression
Ⓐ	NEW	FIN	caseType = A \vee caseType = F
Ⓑ	NEW	CHANGE DIAGN	caseType = B \vee caseType = D
Ⓒ	FIN	END (a_o)	closeCode = H \vee closeCode = X \vee closeCode = BA
Ⓓ	FIN	RELEASE	closeCode \neq H \wedge closeCode \neq X \wedge closeCode \neq BA
Ⓔ	RELEASE	BILLED	caseType = F
Ⓕ	RELEASE	CODE OK	caseType = A \vee caseType = B \vee caseType = D
Ⓖ	REOPEN	CHANGE DIAGN	isClosed = true
Ⓗ	REOPEN	DELETE	isClosed = false

precise than the manually created normative model that we enriched with decision rules in Section 15.2. However, its fitness score is slightly lower since we do not include all the activities in the model to focus on the main behavior and avoid noise affecting the discovery algorithm. Figure 15.7 shows the detailed conformance diagnostics that are returned by the iDHM tool. Despite the good overall fitness of the DC-Net, there are some local conformance issues apparent in Figure 15.7. Many events referring to the activities on the path from *STORNO* to *REOPEN* are diagnosed as *log moves*, i. e., the alignment was not able to match them to an activity execution. Indeed, when comparing the DC-Net with the manually created model in Figure 15.2 it is clear that those activities may also be executed for cases in which *CODE OK* was not executed beforehand. The overall fitness score is not heavily affected since these unmatched events are infrequent.

15.4.3 Comparison with State-of-the-art Techniques

We compared the obtained results with two standard data-unaware process discovery techniques:

- the Inductive Miner⁸⁴ (Figure 15.8) and
- the Heuristics Miner⁸⁵ (Figure 15.9).

The Petri net in Figure 15.8 shows that the Inductive Miner does not succeed in filtering out the noise of the event log. Almost every transition of the Petri net may be repeated due to the loop back to the beginning of the process. Indeed, for some cases the process is repeated. However, activities such as *FIN*, *RELEASE*, and *BILLED* are not shown in the expected order, i. e., the package should first be closed before being release (cf., Figure 15.1).

Probably the exceptional cases in which a single billing package gets reopened and closed several times pose a problem for Inductive Miner. Adjusting the noise

⁸⁴We used the Inductive Miner infrequent with the standard noise filtering parameter, i. e., 0.2.

⁸⁵We simulated the standard Heuristic Miner using the iDHM without considering the conditional dependencies and set $\theta_{obs} = 0$.

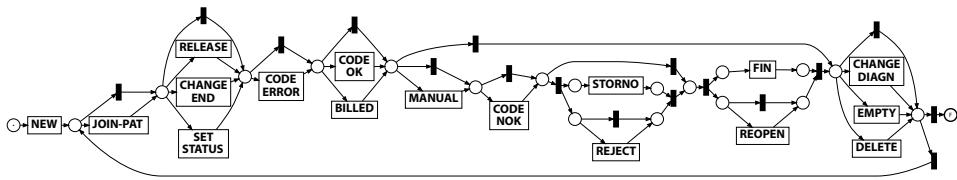


Figure 15.8: Petri net discovered by the Inductive Miner for the hospital billing log. Note that many activities can be skipped and it is possible to loop back. This makes the model very imprecise.

filtering parameter of the Inductive Miner and excluding infrequent activities did not improve the process model.

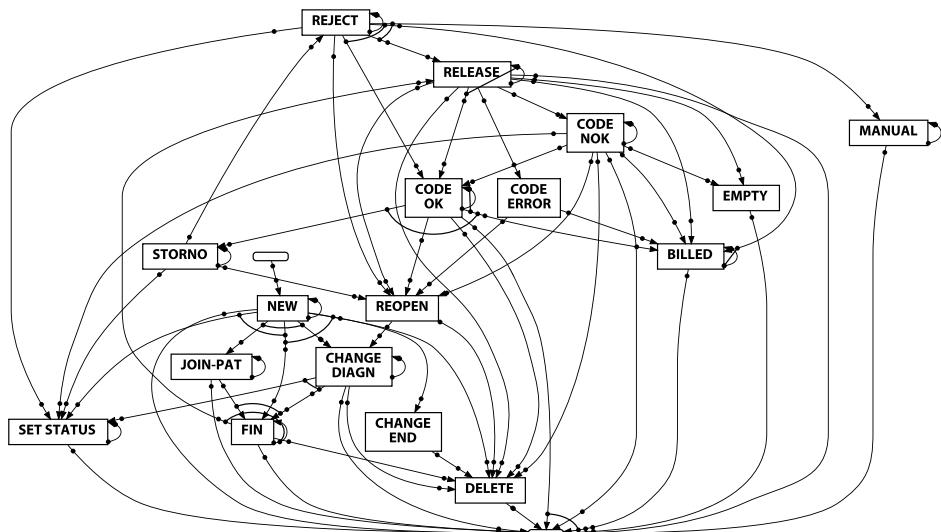


Figure 15.9: C-Net discovered by the standard Heuristic Miner for the hospital billing log.

When using the Heuristics Miner without filtering infrequent behavior (i.e., the standard algorithm as presented in [WR11]), the C-Net in Figure 15.9 is returned. This model shows that including all infrequent behavior leads to an overly complex process model, which is very difficult to understand and, thus, impossible to use in the communication with stakeholders. Conversely, when applying noise filtering (e.g., $\theta_{obs} = 0.1$) then only the main path of the process is shown. The main path of the process is also included in the model returned by the DHM (cf., Figure 15.6). However, some of the conditional behavior revealed by the DHM could not reveal using the standard Heuristics Miner without introducing many more dependencies that we consider as noise. For example, the dependency from RELEASE to

BILLED and the dependency from *REOPEN* to *CHANGE DIAGN* (cf., ③ and ⑦ in Figure 15.6). Moreover, the standard Heuristics Miner does not give insights into the conditions under which dependencies are observed.

15.5 Conclusion

We applied *three of our proposed methods* to the hospital billing process in the context of a case study conducted with a regional Dutch hospital. In this chapter, we described the application of the methods to a random sample of 100,000 cases that were taken from an event log extracted from the ERP system of the hospital.

- In Section 15.2, we discovered several overlapping decision rules that govern the billing process. We validated the discovered rules with stakeholders from the financial department of the hospital. By analyzing the rules, we found that the system manages several types of medical services and that the process differs depending on the type of service. For some types, the billing packages were delete without process or the declaration code was not obtained.
- In Section 15.3, we computed the multi-perspective precision and fitness score of both the model with decision rules and the model without decision rules. The scores show that the decision rules considerably increased the precision of the process model whilst retaining almost the same the fitness score.
- In Section 15.4, in the attempt of improving the results reported in Section 15.2, we discovered a DC-Net that includes both the main behavior of the process as well as five conditional dependencies with our DHM method. We also validated those conditions with stakeholders from the hospital and show that state-of-the-art process discovery techniques have difficulties filtering the noise.

The process stakeholders were primarily interested in projecting the cases onto the normative process model. We found that the system is also used for case types, which do not require handling. However, the underlying reasons for rework and delays could not be explained based on conditions in the data. Overall, three of the proposed methods could be successfully applied and provided valuable insights into the process.

Part V

Closure

16 Conclusion

In this thesis, we proposed five multi-perspective process mining methods that all deal with the interaction of multiple process perspectives. In particular, we looked at the interaction between the following five process perspectives that are often considered in literature [Aal+12; Aal16; BMS16; CKO92; JB96; LA13a; RAH16; Ram17; Ros+11; Sch00]: the control-flow perspective, the resource perspective, the data perspective, the time perspective, and the function perspective. All five proposed methods were implemented and systematically evaluated with synthetic data sets and applied in real-life situations in the context of four case studies.

This last chapter is structured as follows. In Section 16.1, we revisit the contributions of each of the proposed methods and reflect on how they relate to the research goals stated in Section 1.3.1. Several challenges in the field of multi-perspective process mining remain open. We acknowledge these limitations to our work in Section 16.2. We suggest directions for future work in Section 16.3 and conclude this thesis by reflecting on the broader context in Section 16.4.

16.1 Contributions

In Part I of this thesis, we introduced the field of process mining and our view on multi-perspective process mining. As detailed in Section 1.3.1, the goal of our research was threefold: (G1) to develop novel process mining methods that deal with interacting perspectives for all three categories of process mining; (G2) to implement the proposed techniques in efficient and effective tools that can deal with realistic event logs; and (G3) to apply the proposed techniques and tools in real-world scenarios. Next, we revisit the contributions made in the three main parts of this thesis, reflect on how our contributions relate to the goals set, and illustrate what kind of questions can be answered with the proposed methods.

16.1.1 Multi-perspective Conformance (Part II)

Part II of this thesis is concerned with multi-perspective conformance checking, i.e., the diagnosis and quantification of discrepancies between the real execution as recorded by information systems and the desired execution as specified by process models. Our multi-perspective conformance checking technique is grounded in the belief that even though data, resource, and time are separate concerns from a business or modeling perspective, they can be encoded into the data perspective (cf.,

Section 4.2). We assume multi-perspective process models to encode constraints over the data, resource, and time perspective of a process by introducing variables and attaching rules to activities that limit their enablement (i. e., guard expressions). We use the DPN notation (e. g., in Figure 3.4) to model process behavior. All process perspectives are specified in a single integrated model and, thus, we can look at the interaction between multiple perspectives (research goal G1). We proposed two conformance checking methods both of which approach the conformance checking problem from different angles.

Multi-perspective Alignment. In Chapter 5, we presented a method that computes an optimal, multi-perspective, balanced alignment. The alignment relates the behavior modeled in a multi-perspective process model with the behavior observed in an event log. It can be used to calculate the fitness score, i. e., how much of the observed behavior is actually allowed according to the model. We denoted the method as *balanced*, since it balances deviations on the different process perspectives and provides an optimal explanation for the observed behavior in terms of an execution trace of the multi-perspective process model. Deviations that occur on the control-flow perspective (i. e., a misplaced activity) may be explained by wrongly recorded data values. Vice versa, data values that violate constraints may be explained by wrong or missing activity executions. The technique enables to specify statements such as “Skipping activity *Check* is more severe than executing activity *Check* too late” and “Executing activity *Decide* by a different doctor than activity *Visit* is less severe than sending patients with the triage color *Red* to their home”. Moreover, as shown in the road traffic fine management case study (Section 12.2), it allows to answer questions such as “Was the fine *amount* recorded for a case too high” or “Was the activity *Send for Credit Collection* missing in a specific case of the process?”.

Multi-perspective Precision. In Chapter 6, we described a method to measure the precision of a multi-perspective process models with regard to an event log. The precision of a process model can be seen as the fraction of the possible behavior allowed by the model in relation to what has actually been observed, as recorded in the event log. Thus, the precision score is complementary to the fitness score introduced in Chapter 5. Our method is the *first proposal* to measure precision for multi-perspective process models. We *generalized existing precision measures* [Adr+15; MC12] by taking the rules and data values of the multi-perspective process into account. Compared to the state-of-the-art our method is able to answer questions such as “What is the difference in precision between process model A with data rules (i. e., guard expressions) and process model B without data rules?” and “How much more precise is process model C with mutually-exclusive decision rules (i. e., only one of the following activities is activated regardless of the data values) compared to process model D in which some of the decision rule overlap (i. e., multiple

activities are activated for some data values)?"'. We showed that our measure is intuitive by providing several real-life examples. We also used the measure in all the case studies. For example, it is used to evaluate the difference between mutually-exclusive and overlapping rules in the road fines case study (Section 12.3) and in the sepsis case study (Section 13.3).

16.1.2 Multi-perspective Discovery and Enhancement (Part III)

Part III of this thesis is concerned with multi-perspective process discovery and enhancement. The proposed methods leverage the additional information recorded in data attributes (also denoted as event payload) of the event log or use domain knowledge on all process perspectives to discover better process models and enhance existing models. Many existing methods are based on a staged approach, i. e., several process perspectives are discovered separately from each other. Following our first research goal (G1), we proposed two process discovery methods that discover *integrated models* in which multiple perspectives on the process are *intertwined with the control-flow*.

Data-aware Heuristic Process Discovery. In Chapter 8, we presented a method for data-aware heuristic process discovery that aims to reveal infrequent conditional behavior by using recorded data attributes. Data- and control-flow are learned together. The proposed method employs classification techniques to discover conditional dependencies based on the attribute values recorded in the event log. It adds infrequent behavior to the process model such as, e. g., characterized by the following statements "In a few cases patients are assigned a white triage color and leave the hospital" and "Sometimes as a specific nurse reverses the order of the Diagnostics and Visit activity". We implemented the method as Data-aware Heuristic Miner (DHM), which returns the process models as C-Nets. We systematically evaluated the DHM by using a synthetic data set. The experiments showed that it can efficiently handle large event logs with several attributes and distinguish between typical levels of random noise and conditional infrequent behavior. Furthermore, we extended the C-Net notation, which is still focused on the control-flow perspective, to the data-aware DC-Net (cf., Section 8.4) notation. By using the DC-Net notation, we can leverage existing methods for decision mining and conformance checking. The mapping between DC-Nets and DPNs is implemented as part of our iDHM tool. Finally, we applied the DHM in three case studies and show that it does reveal interesting infrequent conditional behavior from real-life event logs.

Guided Multi-perspective Process Discovery. In Chapter 9, we proposed a process discovery method that uses domain knowledge on the functional perspective of the process. The Guided Process Discovery (GPD) method discovers a mapping between low-level events and high-level activities of the process in order to improve

the quality of existing process discovery methods. The method uses *multi-perspective activity patterns* to specify domain knowledge on the function perspective of the process. Activity patterns encode the assumptions on how high-level activities of the process manifest themselves in terms of recorded low-level events. The method employs our multi-perspective alignment method (Chapter 5) to find an optimal mapping between all activity patterns and the low-level event log. Again, we integrate multiple process perspectives in one model. Here, we compute the alignment not for diagnostic purposes but to create an abstracted event log. Based on this abstracted event log, we discover a high-level process model that we validate on the low-level log using an model expansion step. We showed in the evaluation with synthetic data that, despite the computationally expensive search for an optimal alignment, the method can be applied to examples of moderate size and complexity. These examples are often encountered in practice. Finally, we applied the GPD method to each of the four case studies and showed that it can lead to a considerable improvements in the model quality as perceived by stakeholders.

Enhancing Models with Overlapping Decision Rules. In Chapter 10, we described an enhancement method that enriches process models with overlapping decision rules from an event log. Existing techniques only return rules that assume completely deterministic decisions. We observed that this assumption often does not hold. Business rules may be non-deterministic and this ambiguity “*cannot be solved until the business rule is instantiated in a particular situation*” [RW02]. The method builds upon standard classification techniques (we used C4.5 decision trees) and makes an effort to introduce overlap by reclassifying instances that were previously misclassified. To evaluate our technique we used several real-life data sets. We measured the fitness and precision of the process models with discovered rules (cf., the conformance checking technique in Part II). The evaluation showed that our technique is able to produce models with overlapping rules. The discovered models fit the observed behavior better without loosing too much precision. For some decision points, with more than 2 alternative activities, our technique returns rules that are both more fitting and more precise than previous work. Finally, we could discover interpretable overlapping rules in some of the case studies.

16.1.3 Applications (Part IV)

To address our second research goal (G2), we developed two interactive tools: the Multi-perspective Explorer (Section 11.2) and the Interactive Data-aware Heuristic Miner (Section 11.1). Both tools have reached a high level of maturity and have been applied to several case studies in real-world scenarios in three different organizations. Moreover, we tested both tools with large event logs containing more than 5,000,000 events and up to 38 attributes (cf., Chapter 15). To support large event

logs on commodity hardware, we developed the XESLite library⁸⁶, which efficiently stores typical event logs. Next to the two interactive tools, we implemented all proposed methods as plug-ins of the process mining framework ProM.

In Chapters 12 to 15, we presented the results of four of the case studies. The case study presented in Chapter 12 was conducted on an event log provided by a local police force in Italy. We analyzed their process of managing road traffic fines and found that, often, the *Send for Credit Collection* activity was missing for unpaid or underpaid fines. The two case studies presented in Chapters 13 and 14 were conducted in the context of a collaboration with a regional hospital in The Netherlands. We analyzed the trajectories of patients with a sepsis condition and the billing process of the hospital. For the case study presented in Chapter 14 we collaborated with a Norwegian hospital. Here, we analyzed a digital whiteboard system that is used by nurses for their daily work. For each case study, we obtained an event log from the information systems supporting the process under investigation. We applied our proposed methods to the cases and showed that they can be applied in real-life situations and yield valuable insights.

We published anonymized versions of three employed real-life event logs [LM15; Man16a; Man17] in the event log repository of the IEEE Task Force on Process Mining⁸⁷ to enable other researchers to validate our results and use them to test their own methods.

16.2 Limitations

In this section, we acknowledge that there are some limitations to our work. We summarize the most important challenges identified for each of the proposed methods. A more detailed list of limitations for the individual methods is presented in the respective chapters.

Multi-perspective Alignment.

- Comparing modeled and observed behavior is non-trivial and adding multiple integrated process perspectives does increase the complexity of the alignment problem considerably. Our method comes at a increased computational complexity compared to alignment considering only the control-flow since the variable assignment is part of the search space. Therefore, computing an optimal alignment for long traces and complex process models is infeasible. Nevertheless, we could compute the alignment for many processes as encountered in practice. Moreover, it is always possible to filter the event log on the set of activities most relevant.

⁸⁶More information on XESLite can be found in a technical report [Man16b] since we considered the technical challenge to be out of scope for this thesis.

⁸⁷All events logs can be obtained at the repository http://data.4tu.nl/repository/collection:event_logs, which is hosted at the 4TU.centre for research data.

- Our alignment method returns one of several possible optimal alignments. Sometimes, there are multiple possible explanations with the same cost. In some situations it might be desirable to find all optimal alignments. Even though all optimal alignments are associated with the same cost, one of them might provide a better explanation from a domain viewpoint than another one.

Multi-perspective Precision.

- We use alignments to repair the event log to be fully fitting the process model before measuring precision. In extreme cases this can lead to an inherently unreliable precision measurements. For example, in the extreme situation in which the event log is not fitting at all our method still returns a precision value. Therefore, the fitness score should be taken into account together with the precision score.
- Our precision measurement method does not consider the entire behavior of the process model for the measurement of precision. This limitation cannot be avoided when confronted with process models that allow for an infinite amount of behavior.

Data-aware Heuristic Process Discovery.

- Our data-aware heuristic process discovery technique, only considers conditional directly-follows dependencies. More complex patterns of conditional infrequent behavior, e. g., longer sequences or sub-processes, cannot be discovered.
- There is a risk that the C-Nets returned by our method are unsound since it is based on the Heuristics Miner. Unsound models are difficult to interpret. However, it is often possible to look at the subset of process traces that finish properly.

Guided Multi-perspective Process Discovery.

- The proposed GPD method relies on alignments to determine the optimal mapping between low-level events and activities. As mentioned, computing alignments is a computationally expensive operation and may be infeasible for event logs with very long traces and complex abstraction method.
- If a sequence of events fits two activity patterns perfectly when applying the GPD method, one of them will be chosen arbitrarily. This is due to the particularities of the underlying alignment method employed. The cost-based alignment techniques that we use to determine the optimal mapping chooses an arbitrary pattern in case there are multiple mappings with the same cost.

Enhancing Models with Overlapping Decision Rules.

- An inherent limitation of our approach is that it only uses the majority vote to introduce overlapping guards for a decision point with two output transitions. This might cause the guard of one transition to be turned into the rule true, e. g., when the initial guards were based on a single condition. It might also affect the place fitness negatively in case no initial condition was found for the transition.
- Our approach tends to discover guard expressions that are more complex than existing methods. Complex guard expressions may be difficult to interpret by stakeholders.

16.3 Future Work

We summarize the most promising future work related to each of the methods that we have identified while conducting this research. Earlier, we listed more detailed list of future work in Chapters 5, 6 and 8 to 10.

Multi-perspective Alignment.

- Decomposition methods based on the concept of valid decompositions [Aal13] can be used to improve the performance of our multi-perspective conformance checking approach. An initial proposal for a valid decomposition of DPNs was made by de Leoni et al. in [Leo+14a]. However, more efficient decomposition could be developed.
- Sometimes an optimal alignment is not required. An initial *good enough* explanation of the deviations between the observed events and the process model is sufficient to guide the search for conformance problems. Methods to quickly obtain an approximate alignment with quality guarantees are urgently needed. Some initial work on obtaining approximate control-flow alignments has been done in [Don+17; TC16], which might be extendable towards multi-perspective alignments. However, it is difficult to overcome the inherent complexity of the reachability problem, which needs to be solved in order to guarantee that the process projection of the alignment is a valid process trace in the model.
- Capturing the time perspective in constraint requires to encode the execution time of activities in extra variables and define guard expressions with complex calculations (cf., the constraint regarding the Check transition in Figure 3.3). Since time is monotonically increasing within a process instance, it would be possible to pre-compute some of the values in event attributes and simplify the guard expressions. An initial step towards this has been done in [Bal16].

Multi-perspective Precision. It is not straightforward to apply the generic multi-perspective precision measure to variables defined over an infinite domain, e.g., timestamps that are used in constraints on the time perspective or the amount of a loan application. To use such variables together with the introduced measure, the set of possible values would need to be discretized and made finite first. Values could be discretized and made finite by using the values that were actually observed in the event log as guidelines for the values that can be expected.

Data-aware Heuristic Process Discovery.

- The underlying idea of our method, including conditional behavior even if it is infrequent, could be extended from directly-follows relations to more complex patterns of conditional behavior: e.g., conditional long-term dependencies and conditional loops.
- Our method supports the time perspective when it is encoded as data attribute, e.g., conditional relations that appear for cases with a high throughput time. However, the time perspective has particular characteristics that warrant further investigation, e.g., time is monotonically increasing within a process instance and the duration of process activities is usually determined by several correlated events recording life-cycle transitions.

Guided Multi-perspective Process Discovery.

- Work on decomposing or approximating the alignment computation for abstraction models could help to alleviate the performance problems of the method. Since an abstraction model imposes a specific structure on the composition of activity patterns, we believe that a tailor-made decomposition method could be created. This is related to the future work proposed for the balanced alignment method.
- As mentioned, a limitation to our method is that if there are multiple optimal alignments for a sequence of events, i.e., multiple different instantiations of activity patterns could explain the observed behavior, then, one of them will be chosen arbitrarily. A prioritization of activity patterns used during the alignment computation could be introduced. Moreover, it would be possible to introduce a simple heuristic that minimizes the number of pattern instantiations by introducing a small cost for instantiating a pattern to the alignment technique.

Enhancing Models with Overlapping Decision Rules.

- It would be beneficial to investigate the application of other machine-learning techniques to decision mining and decision mining of overlapping rules. It would be useful to have a parameter that influences the expected degree of overlap of the rules to be found. This would enable to steer the discovery of

data-aware process models in the spectrum between fully fitting and fully precision models. A possible realization could be based on rule induction approaches, e. g., based on association rules [AIS93] or rough sets [Ste98]. Rule mining approaches based on rough sets can yield overlapping rules that can be steered in the spectrum from mutually-exclusive (crisp part of the set) towards overlapping (rough part of the set). However, the challenge on how to select appropriate rules from a large set of candidates needs to be met.

- Another important line of future work is to address limitations of decision mining techniques for data sets with imbalanced distributions of classes. Imbalanced distributions are a phenomenon often found in business process (e. g., decisions regarding exceptions or infrequently visited paths of the model). Although we note that our technique is able to reveal rules when one transition is only observed for a small fraction of the cases, a more thorough investigation of this phenomenon is needed.

16.4 Reflection on the Broader Context

We conclude this thesis by reflecting on our contributions in a broader context. The impact of fields such as data mining, machine learning, artificial intelligence, data science, and big data on our society is growing. Due to the growing computing power and storage capacity of today's IT systems, organizations have the opportunity to store information about all their activities. Leveraging knowledge from such recorded data is widely acknowledged to be an important challenge. For example, consider deep learning methods that have been very successful in learning specific complex tasks based on large amounts of data. Process mining is part of this trend towards organizations that are driven by data. Process mining methods operate on event logs that contain traces recorded from the execution of a process. These methods, such as our contributions, can already be useful when applied to smaller amounts of data that is provided in the form of event logs. Often, huge amounts of data are not available in the context of a specific business process. However, process mining requires the input data to be structured in (or convertible to) the form of an event log.

There are many potential benefits by making decisions about the design and optimization of organizational processes more evidence-based, i. e., based on the actual execution of processes as recorded in event logs rather than based on assumptions and feelings of stakeholders. In the light of this, our contributions can be used to get more reliable diagnostics about the process from data (cf., Chapters 5 and 6) and to discover more understandable, complete (i. e., including potentially interesting infrequent process behavior) and balanced (i. e., between a fitting and precise model) process models from data (cf., Chapters 8 to 10).

However, as with any method that leverages recorded data, the results rely on

the quality of the recorded data. Any data-based methods can only be denoted as being “evidence-based” if the recorded data actually represents factual information. For example, in context of our case study on sepsis cases in Section 13.2, we realized that the recorded time stamps of the antibiotics injections were not always representing what happened in reality. Thus, any conclusion drawn upon the results of our methods on unreliable data should be treated with care. However, it does not always need to be the source data that was wrongly recorded. Error may be inadvertently introduced due to mistakes during data preparation or due to errors in the employed software.⁸⁸ Traceability on how the underlying raw data are processed and transparency on how these data are used by the analysis method is required. Moreover, ethical considerations should also be considered when dealing with data that could, possibly, affect the employees and customers of an organization.⁸⁹ To conclude, the responsible usage of data in the context process mining is an important topic. The result of a data-based analysis should always be seen in the context of the real process execution, which might not be 100% reflected in the data that is available.

⁸⁸It is impossible to prove any non-trivial software to be error free. Thus, we can also not be sure that the data transformation during data preparation or the actual analysis was error free.

⁸⁹See, e. g., the Responsible Data Science project <http://www.responsibledatascience.org> for an initial effort into this subject matter.

Bibliography

- [Aa+15] H. van der Aa et al. “On the Fragmentation of Process Information: Challenges, Solutions, and Outlook”. In: *CAiSE 2015 Workshops*. Vol. 214. LNBIP. Springer, 2015, pp. 3–18. doi: [10.1007/978-3-319-19237-6_1](https://doi.org/10.1007/978-3-319-19237-6_1) (page 144).
- [Aa+16] H. van der Aa et al. “Integrated Process and Decision Modeling for Data-driven Processes”. In: *BPM 2015 Workshops*. Vol. 256. LNBIP. Springer, 2016, pp. 405–417. doi: [10.1007/978-3-319-42887-1_33](https://doi.org/10.1007/978-3-319-42887-1_33) (page 175).
- [AAD11] W. M. P. van der Aalst, A. Adriansyah, and B. F. van Dongen. “Causal Nets: A Modeling Language Tailored Towards Process Discovery”. In: *CONCUR 2011*. Vol. 6901. LNCS. Springer, 2011, pp. 28–42. doi: [10.1007/978-3-642-23217-6_3](https://doi.org/10.1007/978-3-642-23217-6_3) (pages 30, 41, 43, 168, 231).
- [AAD12] W. M. P. van der Aalst, A. Adriansyah, and B. F. van Dongen. “Replaying History on Process Models for Conformance Checking and Performance Analysis”. In: *WIREs Data Min Knowl Discovery* 2.2 (2012), pp. 182–192. doi: [10.1002/widm.1045](https://doi.org/10.1002/widm.1045) (pages 6, 55, 56, 58, 106, 114, 132, 133, 203).
- [Aal+10] W. M. P. van der Aalst et al. “Auditing 2.0: Using Process Mining to Support Tomorrow’s Auditor”. In: *IEEE Computer* 43.3 (2010), pp. 90–93. doi: [10.1109/MC.2010.61](https://doi.org/10.1109/MC.2010.61) (page 51).
- [Aal+12] W. M. P. van der Aalst et al. “Process Mining Manifesto”. In: *BPM 2011 Workshops*. Berlin, Heidelberg: Springer, 2012, pp. 169–194. doi: [10.1007/978-3-642-28108-2_19](https://doi.org/10.1007/978-3-642-28108-2_19) (pages 5, 8, 9, 11, 14, 15, 54, 143, 343).
- [Aal13] W. M. P. van der Aalst. “Decomposing Petri Nets for Process Mining: A Generic Approach”. In: *Distrib Parallel Dat* 31.4 (2013), pp. 471–507. doi: [10.1007/s10619-013-7127-5](https://doi.org/10.1007/s10619-013-7127-5) (pages 111, 189, 349).
- [Aal15] W. M. P. van der Aalst. “Extracting Event Data From Databases to Unleash Process Mining”. In: *Management for Professionals*. Springer, 2015, pp. 105–128. doi: [10.1007/978-3-319-14430-6_8](https://doi.org/10.1007/978-3-319-14430-6_8) (page 144).
- [Aal16] W. M. P. van der Aalst. *Process Mining - Data Science in Action, Second Edition*. Springer, 2016. doi: [10.1007/978-3-662-49851-4](https://doi.org/10.1007/978-3-662-49851-4) (pages 6, 7, 11, 12, 24, 25, 29, 41, 43, 45, 52, 54, 113, 139, 140, 143, 147, 185, 187, 188, 308, 343).

- [Aal98] W. M. P. van der Aalst. "The Application of Petri Nets to Workflow Management". In: *Journal of Circuits, Systems and Computers* 08.01 (1998), pp. 21–66. doi: 10.1142/S0218126698000043 (page 34).
- [ABD05] W. M. P. van der Aalst, H. T. de Beer, and B. F. van Dongen. "Process Mining and Verification of Properties: An Approach Based on Temporal Logic". In: *CoopIS 2005*. Vol. 3760. LNCS. Springer, 2005, pp. 130–147. doi: 10.1007/11575771_11 (page 53).
- [ADA11a] A. Adriansyah, B. F. van Dongen, and W. M. P. van der Aalst. "Conformance Checking Using Cost-Based Fitness Analysis". In: *EDOC 2011*. IEEE, 2011, pp. 55–64. doi: 10.1109/EDOC.2011.12 (page 56).
- [ADA11b] A. Adriansyah, B. F. van Dongen, and W. M. P. van der Aalst. "Towards Robust Conformance Checking". In: *BPM 2010 Workshops*. Vol. 66. LNBIP. Springer, 2011, pp. 122–133. doi: 10.1007/978-3-642-20511-8_11 (pages 106, 161).
- [Adr+15] A. Adriansyah et al. "Measuring Precision of Modeled Behavior". In: *Inf Syst E-bus Manag* 13.1 (2015), pp. 37–67. doi: 10.1007/s10257-014-0234-7 (pages 114, 116, 118, 119, 132, 133, 344).
- [Adr14] A. Adriansyah. "Aligning Observed and Modeled Behavior". PhD thesis. Technische Universiteit Eindhoven, 2014. doi: 10.6100/IR770080 (pages 56, 58, 65, 72, 91, 92, 95, 106, 107, 109, 141).
- [ADW08] A. Awad, G. Decker, and M. Weske. "Efficient Compliance Checking Using BPMN-Q and Temporal Logic". In: *BPM 2008*. Vol. 5240. LNCS. Springer, 2008, pp. 326–341. doi: 10.1007/978-3-540-85758-7_24 (pages 51, 107).
- [AGL98] R. Agrawal, D. Gunopulos, and F. Leymann. "Mining Process Models From Workflow Logs". In: *EDBT 1998*. Vol. 1377. LNCS. Springer, 1998, pp. 469–483. doi: 10.1007/bfb0101003 (pages 140, 143).
- [AIS93] R. Agrawal, T. Imieliński, and A. Swami. "Mining Association Rules Between Sets of Items in Large Databases". In: *ACM SIGMOD Record* 22.2 (1993), pp. 207–216. doi: 10.1145/170036.170072 (pages 232, 234, 351).
- [Alb+04] M. Alberti et al. "Specification and Verification of Agent Interactions Using Social Integrity Constraints". In: *Electron Notes Theor Comput Sci* 85.2 (2004). doi: 10.1016/S1571-0661(05)82605-2 (page 108).
- [ALZ15] M. Alizadeh, M. D. Leoni, and N. Zannone. "Constructing Probable Explanations of Nonconformity: A Data-aware and History-based Approach". In: *2015 IEEE Symposium Series on Computational Intelligence*. Institute of Electrical and Electronics Engineers (IEEE), 2015. doi: 10.1109/ssci.2015.194 (page 107).

- [ARS05] W. M. P. van der Aalst, H. A. Reijers, and M. Song. "Discovering Social Networks From Event Logs". In: *Comput Supp Coop Wor* 14.6 (2005), pp. 549–593. doi: 10.1007/s10606-005-9005-9 (page 145).
- [AS11] W. M. P. van der Aalst and C. Stahl. *Modeling Business Processes: A Petri Net Oriented Approach*. MIT press, Cambridge, MA, 2011 (page 176).
- [AS12] R. Accorsi and T. Stocker. "On the Exploitation of Process Mining for Security Audits". In: SAC 2012. ACM, 2012. doi: 10.1145/2245276.2232051 (page 51).
- [Aug+16] A. Augusto et al. "Automated Discovery of Structured Process Models: Discover Structured Vs. Discover and Structure". In: *ER 2016*. Vol. 9974. LNCS. 2016, pp. 313–329. doi: 10.1007/978-3-319-46397-1_25 (pages 177, 243).
- [Aug+17] A. Augusto et al. *Automated Discovery of Process Models from Event Logs: Review and Benchmark*. Tech. rep. 2017 (page 140).
- [Awa10] A. M. H. A. Awad. "A Compliance Management Framework for Business Process Models". PhD thesis. 2010 (page 56).
- [AWM04] W. M. P. van der Aalst, T. Weijters, and L. Maruster. "Workflow Mining: Discovering Process Models From Event Logs". In: *IEEE Trans Knowl Data Eng* 16.9 (2004), pp. 1128–1142. doi: 10.1109/TKDE.2004.47 (pages 140, 174, 241).
- [Bai+15] T. Baier et al. "Matching of Events and Activities: An Approach Based on Behavioral Constraint Satisfaction". In: SAC 2015. ACM, 2015, pp. 1225–1230. doi: 10.1145/2695664.2699491 (page 208).
- [Bai15] T. Baier. "Matching Events and Activities". PhD thesis. Universitat Potsdam, 2015 (pages 144, 181, 208).
- [Bal+15] S. Bala et al. "Mining Project-oriented Business Processes". In: *BPM 2015*. Vol. 9253. LNCS. Springer, 2015, pp. 425–440. doi: 10.1007/978-3-319-23063-4_28 (page 145).
- [Bal16] R. van Balkom. "A Visual Language to Check Conformance of Multi-perspective Process Models against Event Logs. Definition, Implementation, and Evaluation". MA thesis. Technische Universiteit Eindhoven, 2016 (pages 30, 47, 48, 111, 349).
- [BASEL06] *Basel II: International Convergence of Capital Measurement and Capital Standards: A Revised Framework*. 30, 2006. URL: <http://www.bis.org/publ/bcbs128.pdf> (visited on 02/05/2017) (page 51).
- [Bat+15] K. Batoulis et al. "Extracting Decision Logic From Process Models". In: *CAiSE 2015*. Vol. 9097. LNCS. Springer, 2015, pp. 349–366. doi: 10.1007/978-3-319-19069-3_22 (page 175).

- [Baz+17] E. Bazhenova et al. "Discovery of Fuzzy DMN Decision Models From Event Logs". In: *CAiSE 2017*. Vol. 10253. LNCS. Springer, 2017, pp. 629–647. doi: [10.1007/978-3-319-59536-8_39](https://doi.org/10.1007/978-3-319-59536-8_39) (pages 233, 235).
- [BB14] D. Borrego and I. Barba. "Conformance Checking and Diagnosis for Declarative Business Process Models in Data-aware Scenarios". In: *Expert Syst Appl* 41.11 (2014), pp. 5340–5352. doi: [10.1016/j.eswa.2014.03.010](https://doi.org/10.1016/j.eswa.2014.03.010) (pages 56, 108).
- [BBW16] E. Bazhenova, S. Bülow, and M. Weske. "Discovering Decision Models From Event Logs". In: *BIS 2016*. Vol. 255. LNBIP. Springer, 2016, pp. 237–251. doi: [10.1007/978-3-319-39426-8_19](https://doi.org/10.1007/978-3-319-39426-8_19) (pages 145, 175, 213, 231).
- [BDA12] J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst. "A Genetic Algorithm for Discovering Process Trees". In: *CEC 2012*. IEEE, 2012, pp. 1–8. doi: [10.1109/cec.2012.6256458](https://doi.org/10.1109/cec.2012.6256458) (page 174).
- [BDA14] J. C. A. M. Buijs, B. F. van Dongen, and W. M. P. van der Aalst. "Quality Dimensions in Process Discovery: The Importance of Fitness, Precision, Generalization and Simplicity". In: *Int J Coop Inf Syst* 23.1 (2014). doi: [10.1142/S0218843014400012](https://doi.org/10.1142/S0218843014400012) (pages 54, 55, 64, 113, 134).
- [Bec+15] J. Becker et al. "In Search of Information Systems (Grand) Challenges". In: *Bus Inform Syst Eng+* 57.6 (2015), pp. 377–390. doi: [10.1007/s12599-015-0394-0](https://doi.org/10.1007/s12599-015-0394-0) (page 4).
- [Beg17] L. I. Begicheva A.A. "Discovering High-level Process Models From Event Logs". In: *Modeling and Analysis of Information Systems* 24.2 (2017), pp. 125–140. doi: [10.18255/1818-1015-2017-2-125-140](https://doi.org/10.18255/1818-1015-2017-2-125-140) (page 208).
- [Ben08] A. Ben-David. "About the Relationship Between ROC Curves and Cohen's Kappa". In: *Eng Appl Artif Intell* 21.6 (2008), pp. 874–882. doi: [10.1016/j.engappai.2007.09.009](https://doi.org/10.1016/j.engappai.2007.09.009) (page 157).
- [BHM77] S. P. Bradley, A. C. Hax, and T. L. Magnanti. *Applied Mathematical Programming*. Addison-Wesley, 1977 (pages 83, 84, 86).
- [BKR12] S. Bihary, J. Koneti, and S. Roy. "Process Conformance Using CSP". In: *ISEC 2012*. Association for Computing Machinery (ACM), 2012. doi: [10.1145/2134254.2134278](https://doi.org/10.1145/2134254.2134278) (page 106).
- [BLA16] A. Bolt, M. de Leoni, and W. M. P. van der Aalst. "Scientific Workflows for Process Mining: Building Blocks, Scenarios, and Implementation". In: *STTT* 18.6 (2016), pp. 607–628. doi: [10.1007/s10009-015-0399-5](https://doi.org/10.1007/s10009-015-0399-5) (page 203).

- [BLP12] F. Belardinelli, A. Lomuscio, and F. Patrizi. "Verification of GSM-Based Artifact-Centric Systems through Finite Abstraction". In: *IC-SOC 2012*. Vol. 7636. LNCS. Springer, 2012, pp. 17–31. doi: 10.1007/978-3-642-34321-6_2 (pages 107, 108).
- [BMA13] J. C. Bose, R. S. Mans, and W. M. P. van der Aalst. "Wanna Improve Process Mining Results?" In: *CIDM 2013*. IEEE, 2013, pp. 127–134. doi: 10.1109/cidm.2013.6597227 (pages 143, 215).
- [BMS16] A. Burattin, F. M. Maggi, and A. Sperduti. "Conformance Checking Based on Multi-perspective Declarative Process Models". In: *Expert Syst Appl* 65 (15, 2016), pp. 194–211. doi: 10.1016/j.eswa.2016.08.040 (pages 11, 56, 108, 343).
- [BMW14] T. Baier, J. Mendling, and M. Weske. "Bridging Abstraction Layers in Process Mining". In: *Inf Syst* 46 (2014), pp. 123–139. doi: 10.1016/j.is.2014.04.004 (pages 180, 197, 208).
- [Bou+04] M. R. Boutell et al. "Learning Multi-label Scene Classification". In: *Pattern Recogn* 37.9 (2004), pp. 1757–1771. doi: 10.1016/j.patcog.2004.03.009 (page 231).
- [BPEL07] *Web Services Business Process Execution Language Version 2.0*. 2007. URL: <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html> (page 29).
- [BPMN11] *Business Process Model and Notation (BPMN) V2.0*. 2011. URL: <http://www.omg.org/spec/BPMN/2.0/> (pages 6, 46).
- [Bra97] A. P. Bradley. "The Use of the Area under the ROC Curve in the Evaluation of Machine Learning Algorithms". In: *Pattern Recogn* 30.7 (1997), pp. 1145–1159. doi: 10.1016/S0031-3203(96)00142-2 (page 156).
- [BRL16] E. Bellodi, F. Riguzzi, and E. Lamma. "Statistical Relational Learning for Workflow Mining". In: *Intell Data Anal* 20.3 (2016), pp. 515–541. doi: 10.3233/ida-160818 (page 174).
- [Bro+14] S. K. vanden Broucke et al. "Determining Process Model Precision and Generalization With Weighted Artificial Negative Events". In: *IEEE Trans Knowl Data Eng* 26.8 (2014), pp. 1877–1889. doi: 10.1109/tkde.2013.130 (pages 55, 56, 114).
- [Bro14] S. K. L. M. vanden Broucke. "Advances in Process Mining: Artificial Negative Events and Other Techniques". PhD thesis. KU Leuven, 2014 (pages 132, 163, 244).
- [Bui14] J. Buijs. "Flexible Evolutionary Algorithms for Mining Structured Process Models". PhD thesis. Technische Universiteit Eindhoven, 2014. doi: 10.6100/IR780920 (pages 140, 308).

- [Bül+14] S. Bülow et al. "Monitoring of Business Processes With Complex Event Processing". In: *BPM 2013 Workshops*. Vol. 171. LNBIP. Springer, 2014, pp. 277–290. doi: 10.1007/978-3-319-06257-0_22 (page 207).
- [Bus02] N. Busi. "Analysis Issues in Petri Nets With Inhibitor Arcs". In: *Theoret Comput Sci* 275.1-2 (2002), pp. 127–177. doi: 10.1016/s0304-3975(01)00127-x (page 89).
- [BW16] E. Bazhenova and M. Weske. "Deriving Decision Models From Process Models by Enhanced Decision Mining". In: *BPM 2015 Workshops*. Springer, 2016, pp. 444–457. doi: 10.1007/978-3-319-42887-1_36 (page 231).
- [Cal+16] D. Calvanese et al. "Semantics and Analysis of DMN Decision Tables". In: *BPM 2016*. Vol. 9850. LNCS. Springer, 2016, pp. 217–233. doi: 10.1007/978-3-319-45348-4_13 (pages 232, 235).
- [Car12] J. Carmona. "Projection Approaches to Process Mining Using Region-based Techniques". In: *Data Min Knowl Discov* 24.1 (2012), pp. 218–246. doi: 10.1007/s10618-011-0226-x (pages 187, 189).
- [Car13] F. Caron. "Business Process Analytics for Enterprise Risk Management and Auditing". PhD thesis. KU Leuven, 2013, p. 303 (pages 51, 53, 56, 107, 108).
- [Cat+14] S. Catalkaya et al. "Enriching Business Process Models With Decision Rules". In: *BPM 2013 Workshops*. Vol. 171. LNBIP. Springer, 2014, pp. 198–211. doi: 10.1007/978-3-319-06257-0_16 (page 231).
- [CCK08] J. Carmona, J. Cortadella, and M. Kishinevsky. "A Region-based Algorithm for Discovering Petri Nets From Event Logs". In: *BPM 2008*. Vol. 5240. LNCS. Springer, 2008, pp. 358–373. doi: 10.1007/978-3-540-85758-7_26 (pages 140, 174).
- [Che+09] F. Chesani et al. "Exploiting Inductive Logic Programming Techniques for Declarative Process Mining". In: *ToPNoC II*. Vol. 5460. LNCS. Springer, 2009, pp. 278–295. doi: 10.1007/978-3-642-00899-3_16 (page 140).
- [CHM01] J. Cook, C. He, and C. Ma. "Measuring Behavioral Correspondence to a Timed Concurrent Model". In: *ICSM 2001*. IEEE Comput. Soc., 2001, pp. 332–341. doi: 10.1109/ICSM.2001.972746 (page 108).
- [CKO92] B. Curtis, M. I. Kellner, and J. Over. "Process Modeling". In: *Commun Acm* 35.9 (1992), pp. 75–90. doi: 10.1145/130994.130998 (pages 6, 11, 343).
- [CKR13] D. J. Cook, N. C. Krishnan, and P. Rashidi. "Activity Discovery and Activity Recognition: A New Partnership". In: *IEEE Trans Cybern* 43.3 (2013), pp. 820–828. doi: 10.1109/TSMCB.2012.2216873 (page 207).

- [CLH16] R. Conforti, M. La Rosa, and A. H. M. Hofstede. "Filtering Out Infrequent Behavior From Business Process Event Logs". In: *IEEE Trans Knowl Data Eng* (2016). (in press). doi: 10.1109/TKDE.2016.2614680 (pages 144, 174).
- [CM12] G. Cugola and A. Margara. "Processing Flows of Information: From Data Stream to Complex Event Processing". In: *ACM Comput. Surv.* 44.3 (2012), p. 15. doi: 10.1145/2187671.2187677 (pages 206, 207).
- [CM15] C. D. Ciccio and M. Mecella. "On the Discovery of Declarative Control Flows for Artful Processes". In: *ACM Trans Manag Inf Syst* 5.4 (2015), pp. 1–37. doi: 10.1145/2629447 (page 140).
- [Coh60] J. Cohen. "A Coefficient of Agreement for Nominal Scales". In: *Educ Psychol Meas* 20.1 (1960), pp. 37–46. doi: 10.1177/001316446002000104 (pages 156, 157).
- [Con+16] R. Conforti et al. "BPMN Miner: Automated Discovery of BPMN Process Models With Hierarchical Structure". In: *Inf Syst* 56 (2016), pp. 284–303. doi: 10.1016/j.is.2015.07.004 (pages 46, 146).
- [CVB13a] F. Caron, J. Vanthienen, and B. Baesens. "A comprehensive investigation of the applicability of process mining techniques for enterprise risk management". In: *Comput Ind* 64.4 (2013), pp. 464–475. doi: 10.1016/j.compind.2013.02.001 (page 108).
- [CVB13b] F. Caron, J. Vanthienen, and B. Baesens. "Comprehensive Rule-based Compliance Checking and Risk Management With Process Mining". In: *Decis Support Syst* 54.3 (2013), pp. 1357–1369. doi: 10.1016/j.dss.2012.12.012 (pages 56, 107, 108).
- [CW98a] J. E. Cook and A. L. Wolf. "Discovering Models of Software Processes From Event-based Data". In: *ACM Trans Softw Eng Methodol* 7.3 (1998), pp. 215–249. doi: 10.1145/287000.287001 (pages 106, 108, 140).
- [CW98b] J. E. Cook and A. L. Wolf. "Event-based Detection of Concurrency". In: *FSE 1998*. ACM Press, 1998. doi: 10.1145/288195.288214 (page 140).
- [CW99] J. E. Cook and A. L. Wolf. "Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model". In: *ACM T Softw Eng Meth* 8.2 (2 1999), pp. 147–176. doi: 10.1145/304399.304401 (pages 56, 106).
- [Dav93] T. H. Davenport. *Process Innovation: Reengineering Work through Information Technology*. Boston and Mass: Harvard Business School Press, 1993 (page 3).

- [DCC16] B. F. van Dongen, J. Carmona, and T. Chatain. "A Unified Approach for Measuring Precision and Generalization Based on Anti-alignments". In: *BPM 2016*. Vol. 9850. LNCS. Springer, 2016, pp. 39–56. doi: 10.1007/978-3-319-45348-4_3 (page 132).
- [DDG09] R. M. Dijkman, M. Dumas, and L. García-Bañuelos. "Graph Matching Algorithms for Business Process Model Similarity Search". In: *BPM 2009*. Vol. 5701. LNCS. Springer, 2009, pp. 48–63. doi: 10.1007/978-3-642-03848-8_5 (page 172).
- [DDO08] R. M. Dijkman, M. Dumas, and C. Ouyang. "Semantics and Analysis of Business Process Models in BPMN". In: *Inf Softw Technol* 50.12 (2008), pp. 1281–1294. doi: 10.1016/j.infsof.2008.02.006 (page 46).
- [De +16] G. De Giacomo et al. "Computing Trace Alignment against Declarative Process Models through Planning". In: *ICAPS 2016*. AAAI Press, 2016, pp. 367–375 (page 107).
- [De +17a] J. De Smedt et al. "Decision Mining in a Broader Context: An Overview of the Current Landscape and Future Directions". In: *BPM 2016 Workshops*. Vol. 281. LNBI. Springer, 2017, pp. 197–207. doi: 10.1007/978-3-319-58457-7_15 (page 175).
- [De +17b] J. De Smedt et al. "Towards a Holistic Discovery of Decisions in Process-aware Information Systems". In: *BPM 2017*. LNCS. to appear. Springer, 2017 (pages 175, 233, 235).
- [DGP16] C. Diamantini, L. Genga, and D. Potena. "Behavioral Process Mining for Unstructured Processes". In: *J Intell Inf Syst* 47.1 (2016), pp. 5–32. doi: 10.1007/s10844-016-0394-7 (pages 187, 189).
- [DH73] M. Davis and R. Hersh. "Hilbert's 10th Problem". In: *Sci Am* 229.5 (1973), pp. 84–91. doi: 10.1038/scientificamerican1173-84 (page 70).
- [Dij59] E. W. Dijkstra. "A Note on Two Problems in Connexion With Graphs". In: *Numer Math* 1.1 (1959), pp. 269–271. doi: 10.1007/bf01386390 (page 71).
- [DMM14] R. De Masellis, F. M. Maggi, and M. Montali. "Monitoring Data-aware Business Constraints With Finite State Automata". In: *ICSSP 2014*. New York, NY, USA: ACM Press, 2014, pp. 134–143. doi: 10.1145/2600821.2600835 (page 108).
- [DMN16] *Decision Model and Notation (DMN) V1.1*. 2016. url: <http://www.omg.org/spec/DMN/1.1/> (pages 10, 133, 175, 212, 232).
- [Don+17] B. van Dongen et al. "Aligning Modeled and Observed Behavior: A Compromise Between Complexity and Quality". In: *CAiSE 2017*. 2017 (pages 111, 209, 244, 349).

- [DP85] R. Dechter and J. Pearl. "Generalized best-first search strategies and the optimality of A*". In: *J ACM* 32 (3 1985), pp. 505–536. doi: 10.1145/3828.3830 (pages 70, 81, 89).
- [DS90] T. H. Davenport and J. E. Short. "The New Industrial Engineering: Information Technology and Business Process Redesign". In: *Sloan Manage Rev* 31.4 (1990) (page 3).
- [Dum+13] M. Dumas et al. *Fundamentals of Business Process Management*. DOI: 10.1007/978-3-642-33143-5. Springer, 2013 (page 46).
- [Dun+14] R. Dunkl et al. "A Method for Analyzing Time Series Data in Process Mining: Application and Extension of Decision Point Analysis". In: *CAiSE Forum 2014*. Vol. 204. LNBP. Springer, 2014, pp. 68–84. doi: 10.1007/978-3-319-19270-3_5 (page 231).
- [Elg+14] A. Elgammal et al. "Formalizing and Applying Compliance Patterns for Business Process Compliance". In: *Softw Syst Model* 15.1 (2014), pp. 119–146. doi: 10.1007/s10270-014-0395-3 (page 56).
- [ESA16] M. L. van Eck, N. Sidorova, and W. M. P. van der Aalst. "Discovering and Exploring State-based Models for Multi-perspective Processes". In: *BPM 2016*. Vol. 9850. LNCS. Springer, 2016, pp. 142–157. doi: 10.1007/978-3-319-45348-4_9 (page 146).
- [ESP93] CIMOSA: *Open System Architecture for CIM*. Springer, 1993. doi: 10.1007/978-3-642-58064-2 (page 14).
- [FA15] D. Fahland and W. M. van der Aalst. "Model Repair — Aligning Process Models to Reality". In: *Inf Syst* 47 (2015), pp. 220–243. doi: 10.1016/j.is.2013.12.007 (page 141).
- [Faz+15] B. Fazzinga et al. "A Probabilistic Unified Framework for Event Abstraction and Process Detection From Log Data". In: *CoopIS 2015*. Vol. 9415. LNCS. Springer, 2015, pp. 320–328. doi: 10.1007/978-3-319-26148-5_20 (page 208).
- [FGP15] F. Folino, M. Guarascio, and L. Pontieri. "Mining Multi-variant Process Models From Low-level Logs". English. In: *BIS 2015*. Vol. 208. LNBP. Springer, 2015, pp. 165–177. doi: 10.1007/978-3-319-19027-3_14 (page 207).
- [FSR13] D. R. Ferreira, F. Szimanski, and C. G. Ralha. "Mining the Low-level Behaviour of Agents in High-level Business Processes". In: *IJBPM* 6.2 (2013), pp. 146–166. doi: 10.1504/IJBPM.2013.054678 (page 207).
- [FSR14] D. R. Ferreira, F. Szimanski, and C. G. Ralha. "Improving Process Models by Mining Mappings of Low-level Events to High-level Activities". In: *J Intell Inf Syst* 43.2 (2014), pp. 379–407. doi: 10.1007/s10844-014-0327-2 (page 207).

- [GA07] C. W. Günther and W. M. P. van der Aalst. "Fuzzy Mining - Adaptive Process Simplification Based on Multi-perspective Metrics". In: *BPM 2007*. Vol. 4714. LNCS. Springer, 2007, pp. 328–343. doi: 10.1007/978-3-540-75183-0_24 (pages 41, 174, 207, 241).
- [Gar+17] L. García-Bañuelos et al. "Complete and Interpretable Conformance Checking of Business Processes". In: *IEEE Trans Softw Eng* (2017). doi: 10.1109/tse.2017.2668418 (page 107).
- [GCW16] L. George, B. Cadonna, and M. Weidlich. "IL-Miner: Instance-level Discovery of Complex Event Patterns". In: *PVLDB* 10.1 (2016), pp. 25–36. doi: 10.14778/3015270.3015273 (page 207).
- [GGP15] M. T. Gómez-López, R. M. Gasca, and J. M. Pérez-Álvarez. "Compliance Validation and Diagnosis of Business Data Constraints in Business Processes at Runtime". In: *Inf Syst* 48 (2015), pp. 26–43. doi: 10.1016/j.is.2014.07.007 (page 108).
- [Ghi+08] L. Ghionna et al. "Outlier Detection Techniques for Process Mining Applications". In: *ISMIS 2008*. Vol. 4994. LNAI. Springer, 2008, pp. 150–159. doi: 10.1007/978-3-540-68123-6_17 (page 174).
- [GMS06] G. Governatori, Z. Milosevic, and S. W. Sadiq. "Compliance Checking Between Business Processes and Business Contracts". In: *EDOC 2006*. IEEE, 2006, pp. 221–232. doi: 10.1109/edoc.2006.22 (page 107).
- [Goe+09] S. Goedertier et al. "Robust Process Discovery With Artificial Negative Events". In: *J Mach Learn Res* 10 (2009), pp. 1305–1340 (page 174).
- [Goe08] S. Goedertier. "Declarative Techniques for Modeling and Mining Business Processes". PhD thesis. KU Leuven, 2008 (page 140).
- [GRA10] C. W. Günther, A. Rozinat, and W. M. P. van der Aalst. "Activity Mining by Global Trace Segmentation". In: *BPM 2009 Workshops*. Vol. 43. LNBIIP. Springer, 2010, pp. 128–139. doi: 10.1007/978-3-642-12186-9_13 (pages 180, 207).
- [Gre+06] G. Greco et al. "Discovering Expressive Process Models by Clustering Log Traces". In: *IEEE Trans Knowl Data Eng* 18 (8 2006), pp. 1010–1027. doi: 10.1109/TKDE.2006.123 (pages 114, 132).
- [Gri+04] D. Grigori et al. "Business Process Intelligence". In: *Comput Ind* 53.3 (2004), pp. 321–343. doi: <http://dx.doi.org/10.1016/j.compind.2003.10.007> (pages 213, 215, 217, 231).
- [GSP14] J. Ghattas, P. Soffer, and M. Peleg. "Improving Business Process Decision Making Based on Past Experience". In: *Decis Support Syst* 59 (2014), pp. 93–107. doi: <http://dx.doi.org/10.1016/j.dss.2013.10.009> (page 231).

- [Gün09] C. W. Günther. "Process Mining in Flexible Environments". PhD thesis. Technische Universiteit Eindhoven, 2009. doi: 10.6100/IR644335 (page 140).
- [HC93] M. Hammer and J. Champy. *Reengineering the Corporation: A Manifesto for Business Revolution*. 1st ed. New York and NY: HarperBusiness, 1993 (page 3).
- [HMS12] T. Hildebrandt, R. R. Mukkamala, and T. Slaats. "Nested Dynamic Condition Response Graphs". In: *FSEN 2011*. Vol. 7141. LNCS. Springer, 2012, pp. 343–350. doi: 10.1007/978-3-642-29320-7_23 (page 53).
- [HV14] S. Hallé and S. Varvaressos. "A Formalization of Complex Event Stream Processing". In: *EDOC 2014*. IEEE Computer Society, 2014, pp. 2–11. doi: 10.1109/EDOC.2014.12 (page 207).
- [HVA14] B. F. A. Hompes, H. M. W. (Verbeek, and W. M. P. van der Aalst. "Finding Suitable Activity Clusters for Decomposed Process Discovery". In: *SIMPDA 2014*. Vol. 237. LNBP. Springer, 2014, pp. 32–57. doi: 10.1007/978-3-319-27243-6_2 (pages 187, 189).
- [HWG12] J. Hoffmann, I. Weber, and G. Governatori. "On Compliance Checking for Clausal Constraints in Annotated Process Models". English. In: *Inf Syst Front* 14.2 (2012), pp. 155–177. doi: 10.1007/s10796-009-9179-7 (page 107).
- [IEEECIS16] IEEE Computational Intelligence Society. *IEEE Standard for Extensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams*. 2016 (pages 86, 110, 188, 198, 219, 246, 262).
- [ISO15] *ISO 9000: International Standards for Quality Management*. Genève, Switzerland, 2015. URL: http://www.iso.org/iso/home/standards/management-standards/iso_9000.htm (pages 3, 6, 51).
- [JA09] R. P. Jagadeesh Chandra Bose and W. M. P. van der Aalst. "Abstractions in Process Mining: A Taxonomy of Patterns". In: *BPM 2009*. Vol. 5701. LNCS. Springer, 2009, pp. 159–175. doi: 10.1007/978-3-642-03848-8_12 (pages 187, 189, 207).
- [JA12] R. P. Jagadeesh Chandra Bose and W. M. P. van der Aalst. "Process Diagnostics Using Trace Alignment: Opportunities, Issues, and Challenges". In: *Inf Syst* 37.2 (2012), pp. 117–141. doi: 10.1016/j.is.2011.08.003 (page 106).
- [Jan+16] G. Janssenswillen et al. "Measuring the Quality of Models With Respect to the Underlying System: An Empirical Study". In: *BPM 2016*. Vol. 9850. LNCS. Springer, 2016, pp. 73–89. doi: 10.1007/978-3-319-45348-4_5 (pages 55, 134).

- [Jan98] C. Z. Janikow. "Fuzzy Decision Trees: Issues and Methods". In: *IEEE Trans Systems Man and Cybernetics, Part B* 28.1 (1998), pp. 1–14. doi: 10.1109/3477.658573 (page 232).
- [JB96] S. Jablonski and C. Bussler. *Workflow Management - Modeling Concepts, Architecture and Implementation*. International Thomson, 1996 (pages 11, 343).
- [JJ13] W. Jareevongpiboon and P. Janecek. "Ontological Approach to Enhance Results of Business Process Mining and Analysis". In: *Bus Process Manag J* 19.3 (2013), pp. 459–476. doi: 10.1108/14637151311319905 (page 231).
- [JK09] K. Jensen and L. Kristensen. *Coloured Petri Nets*. Springer Verlag, 2009 (page 34).
- [JKW07] K. Jensen, L. M. Kristensen, and L. Wells. "Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems". In: *Int J Software Tools Technol Trans* 9.3-4 (2007), pp. 213–254. doi: 10.1007/s10009-007-0038-x (pages 97, 171).
- [KAL+16a] A. A. Kalenkova et al. *Discovering High-level BPMN Process Models From Event Data*. Tech. rep. BPM Center Report BPM-16-09, BPMcenter.org, 2016 (page 46).
- [KAL+16b] A. A. Kalenkova et al. "Process Mining Using BPMN: Relating Event Logs and Process Models". In: *MODELS 2016*. New York, NY, USA: ACM Press, 2016, pp. 123–123. doi: 10.1145/2976767.2987688 (page 46).
- [KBP12] U. Kaymak, A. Ben-David, and R. Potharst. "The Auk: A Simple Alternative to the AUC". In: *Eng Appl Artif Intell* 25.5 (2012), pp. 1082–1089. doi: 10.1016/j.engappai.2012.02.012 (page 157).
- [Kha+08] M. E. Kharbili et al. "Business Process Compliance Checking: Current State and Future Challenges". In: *MobIS 2008*. Vol. 141. LNI. Bonn: GI, 2008, pp. 107–113 (page 51).
- [KMS07] D. Karagiannis, J. Mylopoulos, and M. Schwab. "Business Process-based Regulation Compliance: The Case of the Sarbanes-oxley Act". In: *RE 2007*. 2007, pp. 315–321. doi: 10.1109/RE.2007.15 (page 51).
- [Knu+10] D. Knuplesch et al. "On Enabling Data-aware Compliance Checking of Business Process Models". In: *ER 2010*. Vol. 6412. LNCS. Springer, 2010, pp. 332–346. doi: 10.1007/978-3-642-16373-9_24 (page 56).
- [KR16] D. Knuplesch and M. Reichert. "A Visual Language for Modeling Multiple Perspectives of Business Process Compliance Rules". In: *Softw Syst Model* (2016). doi: 10.1007/s10270-016-0526-0 (page 47).

- [KRK17] D. Knuplesch, M. Reichert, and A. Kumar. "A Framework for Visually Monitoring Business Process Compliance". In: *Inf Syst* 64 (2017), pp. 381–409. doi: 10.1016/j.is.2016.10.006 (page 47).
- [KSN92] G. Keller, A.-W. Scheer, and M. Nüttgens. *Semantische Prozeßmodellierung Auf Der Grundlage Ereignisgesteuerter Prozeßketten (EPK)*. Tech. rep. 1992 (page 29).
- [KZ99] K. Kosanke and M. Zelm. "CIMOSA Modelling Processes". In: *Comput Ind* 40.2-3 (1999), pp. 141–153. doi: 10.1016/s0166-3615(99)00020-2 (page 14).
- [LA13a] M. de Leoni and W. M. P. van der Aalst. "Aligning Event Logs and Process Models for Multi-perspective Conformance Checking: An Approach Based on Integer Linear Programming". In: *BPM 2013*. Vol. 8094. LNCS. Springer, 2013, pp. 113–129. doi: 10.1007/978-3-642-40176-3_10 (pages 11, 32, 35, 36, 38, 39, 56, 58, 65, 74, 84, 85, 95–98, 101, 104, 105, 109, 110, 266, 267, 287, 343).
- [LA13b] M. de Leoni and W. M. P. van der Aalst. "Data-aware Process Mining: Discovering Decisions in Processes Using Alignments". In: *SAC 2013*. ACM, 2013, pp. 1454–1461. doi: 10.1145/2480362.2480633 (pages 28, 32, 129, 141, 145, 168, 170, 175, 213–215, 217, 218, 221, 225, 226, 231, 233, 245, 275, 299, 330).
- [LA15] M. Leemans and W. M. P. van der Aalst. "Discovery of Frequent Episodes in Event Logs". In: *SIMPDA 2014*. Vol. 237. LNBIP. Springer, 2015, pp. 1–31. doi: 10.1007/978-3-319-27243-6_1 (pages 187, 189, 209).
- [LAD12] M. de Leoni, W. M. P. van der Aalst, and B. F. van Dongen. "Data- and Resource-Aware Conformance Checking of Business Processes". In: *BIS 2012*. Vol. 117. LNBIP. Springer, 2012, pp. 48–59. doi: 10.1007/978-3-642-30359-3_5 (page 108).
- [LDG13] M. de Leoni, M. Dumas, and L. García-Bañuelos. "Discovering Branching Conditions From Business Process Execution Logs". In: *FASE 2013*. Vol. 7793. LNCS. Springer, 2013, pp. 114–129. doi: 10.1007/978-3-642-37057-1_9 (pages 145, 175, 231).
- [Lee17] S. J. Leemans. "Robust Process Mining With Guarantees". PhD thesis. Technische Universiteit Eindhoven, 2017 (page 140).
- [Leo+14a] M. de Leoni et al. "Decomposing Alignment-based Conformance Checking of Data-aware Process Models". In: *CoopIS 2014*. Springer, 2014, pp. 3–20. doi: 10.1007/978-3-662-45563-0_1 (pages 111, 349).
- [Leo+14b] H. Leopold et al. "Simplifying Process Model Abstraction: Techniques for Generating Model Names". In: *Inf Syst* 39 (2014), pp. 134–151. doi: 10.1016/j.is.2013.06.007 (page 189).

- [LFA13] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst. "Discovering Block-structured Process Models From Event Logs - A Constructive Approach". In: *Petri Nets 2013*. Vol. 7927. LNCS. Springer, 2013, pp. 311–329. doi: [10.1007/978-3-642-38697-8_17](https://doi.org/10.1007/978-3-642-38697-8_17) (pages 129, 149, 174).
- [LFA15] X. Lu, D. Fahland, and W. M. P. van der Aalst. "Conformance Checking Based on Partially Ordered Event Data". In: *BPM 2014 Workshops*. Vol. 202. LNBIP. Springer, 2015, pp. 75–88. doi: [10.1007/978-3-319-15895-2_7](https://doi.org/10.1007/978-3-319-15895-2_7) (page 107).
- [LFA16] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst. "Using Life Cycle Information in Process Discovery". In: *BPM 2015 Workshops*. Vol. 256. LNBIP. Springer, 2016, pp. 204–217. doi: [10.1007/978-3-319-42887-1_17](https://doi.org/10.1007/978-3-319-42887-1_17) (pages 188, 200, 308, 319).
- [Liu+16] Y. Liu et al. "From Action to Activity: Sensor-based Activity Recognition". In: *Neurocomputing* 181 (2016), pp. 108–115. doi: [10.1016/j.neucom.2015.08.096](https://doi.org/10.1016/j.neucom.2015.08.096) (pages 206, 207).
- [LK77] J. R. Landis and G. G. Koch. "The Measurement of Observer Agreement for Categorical Data". In: *Biometrics* 33.1 (1977), pp. 159–174 (page 278).
- [LM15] M. de Leoni and F. Mannhardt. *Road Traffic Fine Management Process*. Dataset. 2015 (pages 129, 227, 247, 262, 347).
- [LM17] M. Leoni and A. Marrella. "Aligning Real Process Executions and Prescriptive Process Models through Automated Planning". In: *Expert Syst Appl* (2017). doi: [10.1016/j.eswa.2017.03.047](https://doi.org/10.1016/j.eswa.2017.03.047) (pages 107, 111, 206).
- [LMA12] M. de Leoni, F. M. Maggi, and W. M. P. van der Aalst. "Aligning Event Logs and Declarative Process Models for Conformance Checking". In: *BPM 2012*. Vol. 7481. LNCS. Springer, 2012, pp. 82–97. doi: [10.1007/978-3-642-32885-5_6](https://doi.org/10.1007/978-3-642-32885-5_6) (page 107).
- [LMX07] Y. Liu, S. Müller, and K. Xu. "A Static Compliance-checking Framework for Business Process Models". In: *IBM Syst J* 46.2 (2007), pp. 335–361 (page 107).
- [López+16] M. T. G. López et al. "Computing Alignments With Constraint Programming: The Acyclic Case". In: *ATAED 2016*. Vol. 1592. CEUR Workshop Proceedings. CEUR-WS.org, 2016, pp. 96–110 (pages 107, 111).
- [LR07] R. Lenz and M. Reichert. "IT Support for Healthcare Processes – Premises, Challenges, Perspectives". In: *Data Knowl Eng* 61.1 (2007), pp. 39–58. doi: [10.1016/j.datak.2006.04.007](https://doi.org/10.1016/j.datak.2006.04.007) (page 293).

- [Ly+11] L. T. Ly et al. "Monitoring Business Process Compliance Using Compliance Rule Graphs". In: *CoopIS 2011*. Vol. 7044. LNCS. Springer, 2011, pp. 82–99. doi: [10.1007/978-3-642-25109-2_7](https://doi.org/10.1007/978-3-642-25109-2_7) (pages 53, 107, 108).
- [Ly+15] L. T. Ly et al. "Compliance Monitoring in Business Processes: Functionalities, Application, and Tool-support". In: *Inf Syst* 54 (2015), pp. 209–234. doi: [10.1016/j.is.2015.02.007](https://doi.org/10.1016/j.is.2015.02.007) (page 51).
- [LYC15] V. Liesaputra, S. Yongchareon, and S. Chaisiri. "Efficient Process Model Discovery Using Maximal Pattern Mining". In: *BPM 2105*. Vol. 9253. LNCS. Springer, 2015, pp. 441–456. doi: [10.1007/978-3-319-23063-4_29](https://doi.org/10.1007/978-3-319-23063-4_29) (page 174).
- [M+16] S. M et al. "The third international consensus definitions for sepsis and septic shock (sepsis-3)". In: *JAMA* 315.8 (2016), pp. 801–810. doi: [10.1001/jama.2016.0287](https://doi.org/10.1001/jama.2016.0287) (page 290).
- [Mag+13] F. M. Maggi et al. "Discovering Data-aware Declarative Process Models From Event Logs". In: *BPM 2013*. Vol. 8094. LNCS. Springer, 2013, pp. 81–96. doi: [10.1007/978-3-642-40176-3_8](https://doi.org/10.1007/978-3-642-40176-3_8) (page 146).
- [Man+14] F. Mannhardt et al. *Balanced Multi-Perspective Checking of Process Conformance*. Tech. rep. BPM Center Report BPM-14-07, BPMcenter.org, 2014 (pages 49, 68, 261).
- [Man+16a] F. Mannhardt et al. "Balanced Multi-perspective Checking of Process Conformance". In: *Computing* 98.4 (2016), pp. 407–437. doi: [10.1007/s00607-015-0441-1](https://doi.org/10.1007/s00607-015-0441-1) (pages 49, 68, 96, 97, 129, 131, 227, 265).
- [Man+16b] F. Mannhardt et al. "Decision Mining Revisited - Discovering Overlapping Rules". In: *CAiSE 2016*. Vol. 9694. LNCS. Springer, 2016, pp. 377–392. doi: [10.1007/978-3-319-39696-5_23](https://doi.org/10.1007/978-3-319-39696-5_23) (pages 137, 217, 233, 289).
- [Man+16c] F. Mannhardt et al. "From Low-level Events to Activities - A Pattern-based Approach". In: *BPM 2016*. Vol. 9850. LNCS. Springer, 2016, pp. 125–141. doi: [10.1007/978-3-319-45348-4_8](https://doi.org/10.1007/978-3-319-45348-4_8) (pages 137, 186, 203, 313).
- [Man+16d] F. Mannhardt et al. "Measuring the Precision of Multi-perspective Process Models". In: *BPM 2015 Workshops*. Vol. 256. LNBP. Springer, 2016, pp. 113–125. doi: [10.1007/978-3-319-42887-1_10](https://doi.org/10.1007/978-3-319-42887-1_10) (pages 49, 116, 129, 133).
- [Man+17] F. Mannhardt et al. "Data-driven Process Discovery - Revealing Conditional Infrequent Behavior From Event Logs". In: *CAiSE 2017*. Vol. 10253. LNCS. 2017, pp. 545–560. doi: [10.1007/978-3-319-59536-8_34](https://doi.org/10.1007/978-3-319-59536-8_34) (pages 137, 325).

- [Man+18] F. Mannhardt et al. "Guided Process Discovery - A Pattern-based Approach". In: *Inf Syst* (2018). submitted (page 137).
- [Man16a] F. Mannhardt. *Sepsis Cases - Event Log. Eindhoven University of Technology. Dataset*. 2016. doi: 10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460 (pages 227, 228, 294, 347).
- [Man16b] F. Mannhardt. *XESLite - Managing Large XES Event Logs in ProM*. BPM Center Report BPM-16-04. BPMCenter.org, 2016. URL: <http://bpmcenter.org/wp-content/uploads/reports/2016/BPM-16-04.pdf> (pages 246, 347).
- [Man17] F. Mannhardt. *Hospital Billing - Event Log. Eindhoven University of Technology. Dataset*. 2017. doi: 10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfcb741 (pages 326, 347).
- [Mar03] I. Maros. *Computational Techniques of the Simplex Method*. Springer US, 2003. doi: 10.1007/978-1-4615-0257-9 (page 87).
- [MAW08] A. A. de Medeiros, W. van der Aalst, and A. Weijters. "Quantifying Process Equivalence Based on Observed Behavior". In: *Data Knowl Eng* 64.1 (2008), pp. 55–74. doi: 10.1016/j.datak.2007.06.010 (pages 56, 106).
- [MB17] F. Mannhardt and D. Blinde. "Analyzing the Trajectories of Patients With Sepsis Using Process Mining". In: *RADAR+EMISA 2017. Vol. 1859. CEUR Workshop Proceedings*. CEUR-WS.org, 2017, pp. 72–80 (pages 227, 289).
- [MC10] J. Munoz-Gama and J. Carmona. "A Fresh Look at Precision in Process Conformance". In: *BPM 2010. Vol. 6336. LNCS*. Springer, 2010, pp. 211–226. doi: 10.1007/978-3-642-15618-2_16 (pages 56, 113).
- [MC12] J. Munoz-Gama and J. Carmona. "A General Framework for Precision Checking". In: *Int J Innov Comput I 8.7(B)* (2012), pp. 5317–5339 (pages 114, 118, 132–134, 344).
- [MCA14] J. Munoz-Gama, J. Carmona, and W. M. van der Aalst. "Single-entry Single-exit Decomposed Conformance Checking". In: *Inf Syst* 46 (2014), pp. 102–122. doi: 10.1016/j.is.2014.04.003 (pages 107, 206).
- [Med06] A. A. de Medeiros. "Genetic Process Mining". PhD thesis. Eindhoven: Eindhoven University of Technology, 2006 (page 140).
- [Men08] J. Mendling. *Metrics for Process Models*. Vol. 6. LNBIP. Springer, 2008. doi: 10.1007/978-3-540-89224-3 (page 54).
- [Min67] M. L. Minsky. *Computation: Finite and Infinite Machines (automatic Computation)*. Prentice Hall, 1967 (page 89).

- [MLR15a] F. Mannhardt, M. de Leoni, and H. A. Reijers. "Extending Process Logs With Events From Supplementary Sources". In: *BPM 2015 Workshops*. Vol. 202. LNBIP. Springer, 2015, pp. 235–247. doi: 10.1007/978-3-319-15895-2_21 (page 144).
- [MLR15b] F. Mannhardt, M. de Leoni, and H. A. Reijers. "The Multi-perspective Process Explorer". In: *BPM 2015 Demos*. Vol. 1418. CEUR Workshop Proceedings. CEUR-WS.org, 2015, pp. 130–134 (pages 225, 246).
- [MLR17] F. Mannhardt, M. de Leoni, and H. A. Reijers. "Heuristic Mining Revamped: An Interactive Data-aware and Conformance-aware Miner". In: *BPM 2017 Demos*. CEUR Workshop Proceedings. CEUR-WS.org, 2017 (page 239).
- [Mol+14] T. Molka et al. "Conformance Checking for BPMN-based Process Models". In: *SAC 2014*. Association for Computing Machinery (ACM), 2014. doi: 10.1145/2554850.2555061 (page 107).
- [Mon10] M. Montali. *Specification and Verification of Declarative Open Interaction Models - A Logic-based Approach*. Vol. 56. LNBIP. Springer, 2010, pp. 1–383. doi: 10.1007/978-3-642-14538-4 (page 108).
- [MR08] M. z. Muehlen and J. Recker. "How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation". In: *CAiSE 2008*. Vol. 5074. LNCS. Springer, 2008, pp. 465–479. doi: 10.1007/978-3-540-69534-9_35 (page 46).
- [MT17] F. Mannhardt and N. Tax. "Unsupervised Event Abstraction Using Pattern Abstraction and Local Process Models". In: *RADAR+EMISA 2017*. Vol. 1859. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 55–63 (pages 189, 209).
- [Mue04] M. zur Muehlen. *Workflow-based Process Controlling. Foundation, Design, and Application of Workflow-driven Process Information Systems*. Vol. 6. Advances in Information Systems and Management Science. Logos Verlag, 2004, p. 315 (page 4).
- [Mur89] T. Murata. "Petri Nets: Properties, Analysis and Applications". In: *Proc IEEE* 77.4 (1989), pp. 541–580. doi: 10.1109/5.24143 (pages 34, 89, 203).
- [MWA07] A. K. A. de Medeiros, A. J. M. M. Weijters, and W. M. P. van der Aalst. "Genetic Process Mining: An Experimental Evaluation". In: *Data Min Knowl Discov* 14.2 (2 2007), pp. 245–304 (pages 106, 114, 118, 132, 174).
- [Nau03] D. D. Nauck. "Fuzzy Data Analysis With NEFCLASS". In: *Internat J Approx Reason* 32.2 (2003), pp. 103–130. doi: 10.1016/S0888-613X(02)00079-8 (page 232).

- [NW70] S. B. Needleman and C. D. Wunsch. "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins". In: *J Mol Biol* 48.3 (1970), pp. 443–453. doi: 10.1016/0022-2836(70)90057-4 (page 106).
- [Oli+13] C. A. L. Oliveira et al. "Reducing the Gap Between Business and Information Systems through Complex Event Processing". In: *Comput Inform* 32.2 (2013), pp. 225–250 (page 207).
- [PCB15] H. Ponce de León, J. Carmona, and S. K. L. M. vanden Broucke. "Incorporating Negative Information in Process Discovery". In: *BPM 2015*. Vol. 9253. LNCS. Springer, 2015, pp. 126–143. doi: 10.1007/978-3-319-23063-4_8 (page 174).
- [Pet62] C. A. Petri. "Kommunikation Mit Automaten". PhD thesis. Universität Hamburg, 1962 (page 29).
- [PFD15] V. Popova, D. Fahland, and M. Dumas. "Artifact Lifecycle Discovery". In: *Int J Coop Inf Syst* 24.01 (2015), p. 1550001. doi: 10.1142/s021884301550001x (page 175).
- [Pol12] A. Polyvyanyy. "Structuring Process Models". PhD thesis. Universität Potsdam, 2012 (page 54).
- [Pow11] D. Powers. "Evaluation: From Precision, Recall and F-measure to ROC, Informedness, Markedness & Correlation". In: *Journal of Machine Learning Technologies* 2 (1 2011) (page 157).
- [PSA07] M. Pesic, H. Schonenberg, and W. M. P. v. d. Aalst. "DECLARE: Full Support for Loosely-structured Processes". In: *EDOC 2007*. 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007). IEEE, 2007, pp. 287–287. doi: 10.1109/EDOC.2007.14 (pages 29, 53).
- [Qui93] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993 (pages 27, 28, 151, 155, 214, 215, 231).
- [RA06] A. Rozinat and W. M. P. van der Aalst. "Decision Mining in ProM". In: *BPM 2006*. Vol. 4102. LNCS. Springer, 2006, pp. 420–425. doi: 10.1007/11841760_33 (pages 141, 145, 211, 213–215, 217, 231).
- [RA08] A. Rozinat and W. M. P. van der Aalst. "Conformance Checking of Processes Based on Monitoring Real Behavior". In: *Inf Syst* 33 (1 2008), pp. 64–95. doi: 10.1016/j.is.2007.07.001 (pages 56, 106, 114, 116, 118, 132).
- [RAH16] N. Russell, W. M. P. V. der Aalst, and A. H. M. ter Hofstede. *Workflow Patterns*. MIT Press Ltd, 8, 2016. 384 pp. (pages 11, 343).
- [Ram17] E. Ramezani. "Understanding Non-compliance". PhD thesis. Technische Universiteit Eindhoven, 2017 (pages 11, 56, 107, 343).

- [Rea+11] J. Read et al. "Classifier Chains for Multi-label Classification". In: *Mach Learn* 85.3 (2011), pp. 333–359. doi: 10.1007/s10994-011-5256-5 (page 232).
- [Rec10] J. Recker. "Opportunities and Constraints: The Current Struggle With BPMN". In: *Bus Process Manag J* 16.1 (2010), pp. 181–201. doi: 10.1108/14637151011018001 (page 46).
- [Red+14] D. Redlich et al. "Constructs Competition Miner: Process Control-flow Discovery of BP-domain Constructs". In: *BPM 2014*. Vol. 8659. LNCS. Springer, 2014, pp. 134–150. doi: 10.1007/978-3-319-10172-9_9 (pages 140, 174).
- [Rei85] W. Reisig. *Petri Nets: An Introduction*. Vol. 4. EATCS Monographs on Theoretical Computer Science. Springer, 1985. doi: 10.1007/978-3-642-69968-9 (page 29).
- [Rem+13] A. J. Rembert et al. "Process Discovery Using Prior Knowledge". In: *ICSOB 2013*. Vol. 8274. LNCS. Springer, 2013, pp. 328–342. doi: 10.1007/978-3-642-45005-1_23 (page 174).
- [Rey+06] P. Reynolds et al. "Pip: Detecting the Unexpected in Distributed Systems". In: *NSDI 2006*. USENIX Association, 2006, pp. 115–128 (page 108).
- [RF12] Á. Rebuge and D. R. Ferreira. "Business Process Analysis in Healthcare Environments: A Methodology Based on Process Mining". In: *Inf Syst* 37.2 (2012), pp. 99–116. doi: 10.1016/j.is.2011.01.003 (page 290).
- [RFA12] E. Ramezani, D. Fahland, and W. M. P. v. d. Aalst. "Where Did I Misbehave? Diagnostic Information in Compliance Checking". In: *BPM 2012*. Vol. 7481. LNCS. Springer, 3, 2012, pp. 262–278. doi: 10.1007/978-3-642-32885-5_21 (pages 53, 56).
- [Rho+17] A. Rhodes et al. "Surviving Sepsis Campaign: International Guidelines for Management of Sepsis and Septic Shock: 2016". In: *Intensive Care Med* 43.3 (2017), pp. 304–377. doi: 10.1007/s00134-017-4683-6 (pages 289, 290, 292, 294, 298).
- [RMD11] H. A. Reijers, J. Mendling, and R. M. Dijkman. "Human and Automatic Modularizations of Process Models to Enhance Their Comprehension". In: *Inf Syst* 36.5 (2011), pp. 881–897. doi: 10.1016/j.is.2011.03.003 (page 181).
- [Rog+13] A. Rogge-Solti et al. "Repairing Event Logs Using Timed Process Models". In: *OTM 2013 Workshops*. Vol. 8186. Springer, 2013, pp. 705–708. doi: 10.1007/978-3-642-41033-8_89 (page 199).

- [Rol98] C. Rolland. "A Comprehensive View of Process Engineering". In: *CAiSE 1998*. Vol. 1413. LNCS. Springer, 1998, pp. 1–24. doi: 10.1007/bfb0054216 (page 6).
- [Ros+11] M. L. Rosa et al. "Configurable Multi-perspective Business Process Models". In: *Inf Syst* 36.2 (2011), pp. 313–340. doi: 10.1016/j.is.2010.07.001 (pages 11, 343).
- [Roz+08] A. Rozinat et al. "The Need for a Process Mining Evaluation Framework in Research and Practice". In: *BPM 2007 Workshops*. Vol. 4928. LNCS. Springer, 2008, pp. 84–89. doi: 10.1007/978-3-540-78238-4_10 (page 54).
- [Roz+09] A. Rozinat et al. "Discovering Simulation Models". In: *Inf Syst* 34.3 (2009), pp. 305–327. doi: 10.1016/j.is.2008.09.002 (pages 141, 175, 213, 215).
- [Roz10] A. Rozinat. "Process Mining: Conformance and Extension". PhD thesis. Eindhoven: Eindhoven University of Technology, 2010 (pages 56, 146, 244).
- [Rus+05] N. Russell et al. "Workflow Resource Patterns: Identification, Representation and Tool Support". In: *CAiSE 2005*. Vol. 3520. LNCS. Springer, 2005, pp. 216–232. doi: 10.1007/11431855_16 (page 57).
- [RVA08] A. Rozinat, M. Veloso, and W. M. P. van der Aalst. *Using Hidden Markov Models to Evaluate the Quality of Discovered Process Models*. BPM Center Report BPM-08-10. 2008 (page 106).
- [RW02] D. Rosca and C. Wild. "Towards a Flexible Deployment of Business Rules". In: *Expert Syst Appl* 23.4 (2002), pp. 385–394. doi: 10.1016/s0957-4174(02)00074-x (pages 215, 346).
- [SA07] M. Song and W. M. van der Aalst. "Supporting Process Mining by Showing Events at a Glance". In: *WITS 2007*. 2007, pp. 139–145 (page 319).
- [Sch+16a] S. Schönig et al. "A Framework for Efficiently Mining the Organisational Perspective of Business Processes". In: *Decis Support Syst* 89 (2016), pp. 87–97. doi: 10.1016/j.dss.2016.06.012 (page 146).
- [Sch+16b] S. Schönig et al. "Discovery of Multi-perspective Declarative Process Models". In: *ICSOC 2016*. Vol. 9936. LNCS. Springer, 2016, pp. 87–103. doi: 10.1007/978-3-319-46295-0_6 (pages 146, 175).
- [Sch+16c] S. Schönig et al. "Efficient and Customisable Declarative Process Mining With SQL". In: *CAiSE 2016*. Vol. 9694. LNCS. Springer, 2016, pp. 290–305. doi: 10.1007/978-3-319-39696-5_18 (page 140).
- [Sch00] A.-W. Scheer. *ARIS — Business Process Modeling*. Springer, 2000. doi: 10.1007/978-3-642-57108-4 (pages 11, 13, 343).

- [Sch86] A. Schrijver. *Theory of Linear and Integer Programming (wiley Series in Discrete Mathematics and Optimization)*. Wiley, 1986 (page 83).
- [Sch92] A.-W. Scheer. *Architecture of Integrated Information Systems*. Springer, 1992. doi: 10.1007/978-3-642-97389-5 (page 13).
- [Sen+14] A. Senderovich et al. "Queue Mining – Predicting Delays in Service Processes". In: *CAiSE 2014*. Vol. 8484. LNCS. Springer, 2014, pp. 42–57. doi: 10.1007/978-3-319-07881-6_4 (page 141).
- [Sen+15] A. Senderovich et al. "Queue Mining for Delay Prediction in Multi-class Service Processes". In: *Inf Syst* 53 (2015), pp. 278–295. doi: 10.1016/j.is.2015.03.010 (page 145).
- [Sen+16a] A. Senderovich et al. "Conformance Checking and Performance Improvement in Scheduled Processes: A Queueing-network Perspective". In: *Inf Syst* 62 (2016), pp. 185–206. doi: 10.1016/j.is.2016.01.002 (page 108).
- [Sen+16b] A. Senderovich et al. "The ROAD From Sensor Data to Process Instances Via Interaction Mining". In: *CAiSE 2016*. Vol. 9694. LNCS. Springer, 2016, pp. 257–273. doi: 10.1007/978-3-319-39696-5_16 (page 207).
- [SGN07] S. W. Sadiq, G. Governatori, and K. Namiri. "Modeling Control Objectives for Business Process Compliance". In: *BPM 2007*. Vol. 4714. LNCS. Springer, 2007, pp. 149–164. doi: 10.1007/978-3-540-75183-0_12 (page 56).
- [SOX02] One Hundred Seventh Congress of the United States of America. *Sarbanes-Oxley Act of 2002*. legislation. 2002. URL: <https://www.congress.gov/bill/107th-congress/house-bill/3763> (visited on 02/05/2017) (page 51).
- [SST11] N. Sidorova, C. Stahl, and N. Trčka. "Soundness Verification for Conceptual Workflow Nets With Data: Early Detection of Errors With the Most Precision Possible". In: *Inf Syst* 36.7 (2011), pp. 1026–1043. doi: 10.1016/j.is.2011.04.004 (page 32).
- [Ste98] J. Stefanowski. "On Rough Set Based Approaches to Induction of Decision Rules". In: *Rough Sets in Knowledge Discovery Vol 1*. 1. Physica Verlag, 1998, pp. 500–529 (pages 232, 234, 351).
- [Sur+17] S. Suriadi et al. "Event Log Imperfection Patterns for Process Mining: Towards a Systematic Approach to Cleaning Event Logs". In: *Inf Syst* 64 (2017), pp. 132–150. doi: 10.1016/j.is.2016.07.011 (pages 144, 147).
- [SW81] T. Smith and M. Waterman. "Identification of Common Molecular Subsequences". In: *J Mol Biol* 147.1 (1981), pp. 195–197. doi: 10.1016/0022-2836(81)90087-5 (page 106).

- [Tax+16a] N. Tax et al. "Mining Local Process Models". In: *Journal of Innovation in Digital Ecosystems* 3.2 (2016). , in press, pp. 183–196. doi: 10.1016/j.jides.2016.11.001 (pages 187, 189, 209).
- [Tax+16b] N. Tax et al. "Event Abstraction for Process Mining Using Supervised Learning Techniques". In: *IntelliSys 2016*. pre-print, <https://arxiv.org/abs/1606.07283>. IEEE, 2016, pp. 161–170 (page 207).
- [TC16] F. Taymouri and J. Carmona. "A Recursive Paradigm for Aligning Observed Behavior of Large Structured Process Models". In: *BPM 2016*. Vol. 9850. LNCS. Springer, 2016, pp. 197–214. doi: 10.1007/978-3-319-45348-4_12 (pages 107, 111, 209, 244, 349).
- [TK07] G. Tsoumakas and I. Katakis. "Multi-label Classification: An Overview". In: *Int J Data Warehous* 2007 (2007), pp. 1–13. doi: 10.4018/978-1-59904-951-9.ch006 (page 231).
- [Tur+12] O. Turetken et al. "Capturing Compliance Requirements: A Pattern-based Approach". In: *IEEE Softw* 29.3 (2012), pp. 28–36. doi: 10.1109/MS.2012.45 (page 56).
- [Tur37] A. M. Turing. "On Computable Numbers, With an Application to the Entscheidungsproblem". In: *Proceedings of the London Mathematical Society* s2-42.1 (1937), pp. 230–265. doi: 10.1112/plms/s2-42.1.230 (page 90).
- [VWC96] J. Vanthienen, G. Wets, and G. Chen. "Incorporating Fuzziness in the Classical Decision Table Formalism". In: *Int J Intell Syst* 11.11 (1996), pp. 879–891. doi: 10.1002/(SICI)1098-111X(199611)11:11<879::AID-INT2>3.0.CO;2-W (page 233).
- [WA03] A. J. M. M. Weijters and W. M. P. van der Aalst. "Rediscovering Workflow Models From Event-based Data Using Little Thumb". In: *Integr Comput Aid E* 10.2 (2003), pp. 151–162 (page 161).
- [WAA06] A. J. M. M. Weijters, W. M. P. van der Aalst, and A. K. Alves de Medeiros. *Process Mining With the Heuristics Miner-algorithm*. Tech. rep. BETA Working Paper Series, WP 166. Eindhoven University of Technology, Eindhoven, 2006 (pages 41, 106, 161).
- [WBC14] J. D. Weerdt, S. K. L. M. vanden Broucke, and F. Caron. "Bidimensional Process Discovery for Mining BPMN Models". In: *BPM 2014 Workshops*. Vol. 202. LNBIP. Springer, 7, 2014, pp. 529–540. doi: 10.1007/978-3-319-15895-2_45 (page 46).
- [Wee+11] J. Weerdt et al. "A Robust F-measure for Evaluating Discovered Process Models". In: *CIDM 2011*. Paris, France: IEEE, 2011, pp. 148–155. doi: 10.1109/cidm.2011.5949428 (page 132).

- [Wee+12] J. D. Weerdt et al. “A Multi-dimensional Quality Assessment of State-of-the-art Process Discovery Algorithms Using Real-life Event Logs”. In: *Inf Syst* 37.7 (2012), pp. 654–676. doi: 10.1016/j.is.2012.02.004 (pages 140, 143, 174).
- [Wei+11] M. Weidlich et al. “Process Compliance Analysis Based on Behavioural Profiles”. In: *Inf Syst* 36.7 (2011), pp. 1009–1025. doi: 10.1016/j.is.2011.04.002 (pages 56, 106).
- [Wei+14] M. Weidlich et al. “Optimizing Event Pattern Matching Using Business Process Models”. In: *IEEE Trans Knowl Data Eng* 26.11 (2014), pp. 2759–2773. doi: 10.1109/TKDE.2014.2302306 (page 207).
- [Wen+07] L. Wen et al. “Mining Process Models With Non-free-choice Constructs”. In: *Data Min Knowl Discov* 15.2 (2007), pp. 145–180. doi: 10.1007/s10618-007-0065-y (page 140).
- [Wen+10] L. Wen et al. “Mining Process Models With Prime Invisible Tasks”. In: *Data Knowl Eng* 69.10 (2010), pp. 999–1021. doi: 10.1016/j.datak.2010.06.001 (page 140).
- [Wer+09] J. M. E. M. van der Werf et al. “Process Discovery Using Integer Linear Programming”. In: *Fundam Inform* 94.3-4 (2009), pp. 387–412. doi: 10.3233/FI-2009-136 (page 140).
- [Won+09] H. J. Wong et al. “Electronic Inpatient Whiteboards: Improving Multidisciplinary Communication and Coordination of Care”. In: *Int J Med Inform* 78.4 (2009), pp. 239–247. doi: 10.1016/j.ijmedinf.2008.07.012 (page 313).
- [WR11] A. J. M. M. Weijters and J. T. S. Ribeiro. “Flexible Heuristics Miner (FHM)”. In: *CIDM 2011*. IEEE, 2011, pp. 310–317. doi: 10.1109/cidm.2011.5949453 (pages 41, 140, 149–151, 154, 159, 161–163, 171, 174, 176, 241, 243, 302, 308, 339).
- [Xu+09] W. Xu et al. “Detecting Large-scale System Problems by Mining Console Logs”. In: *SOSP 2009*. ACM Press, 2009, pp. 117–132. doi: 10.1145/1629575.1629587 (page 108).
- [Zac87] J. A. Zachman. “A Framework for Information Systems Architecture”. In: *IBM Syst J* 26.3 (1987), pp. 276–292. doi: 10.1147/sj.263.0276 (page 14).
- [Zad65] L. Zadeh. “Fuzzy Sets”. In: *Information and Control* 8.3 (1965), pp. 338–353. doi: 10.1016/S0019-9958(65)90241-X (page 232).
- [ZDA15] S. J. van Zelst, B. F. van Dongen, and W. M. P. van der Aalst. “Avoiding Over-fitting in ILP-based Process Discovery”. In: *BPM 2015*. Vol. 9253. LNCS. Springer, 2015, pp. 163–171. doi: 10.1007/978-3-319-23063-4_10 (page 308).

Index

- Alignment, 58, 60
 - Correct synchronous move (\bullet), 59
 - Cost function, 62
 - Decidability, 90
 - Incorrect synchronous move (\otimes), 59
 - Legal moves ($\Gamma_{L,LTS}$), 59
 - Log move (Θ), 59
 - Log projection ($proj_L$), 59
 - Model move (Θ), 59
 - Move, 59
 - Optimal alignment, 63
 - Process projection ($proj_P$), 59
 - Standard cost (κ_1), 63
- Balanced alignment
 - Augmentation function, 74
 - Control-flow successors, 73
 - Search space (Z), 70
- Boolean expression (EXPR), 23
 - Evaluation function ($eval$), 24
- Causal Net
 - Binding sequence (B), 43
 - C-Net (C), 41
 - Dependency relations (D), 42
 - Input bindings (I), 42
 - Language, 45
 - Output bindings (O), 42
 - State function ($state_C$), 44
 - Trace set (TS_C), 45
 - Valid binding sequence, 44
- Cohen's kappa, 157
- Conditional dependency measure, 154
- Conditional directly follows relation, 152
- Data Causal Net
- Binding sequence, 165
- DC-Net, 163
- State, 165
- Trace set, 167
- Valid binding sequences (BS_{DC}), 166
- Data Petri Net
 - Control-flow copy (LN'), 73
 - DPN (N), 35
 - Guard function (gd), 35
 - Initial value (in), 35
 - Invisible transition, 40
 - Labeled DPN (LN), 40
 - Possible process traces, 39
 - Prime variables, 34
 - Process trace, 39
 - Relaxed soundness, 69
 - State, 36
 - Trace set, 39
 - Transition firings, 38
 - Turing complete, 89
 - Variable domain (dom), 35
 - Variables (V_P), 35
 - Write function (wr), 35
- Data-aware Heuristic Miner
 - Binding threshold (θ_{bin}), 158
 - Condition threshold (θ_{con}), 157
 - Dependency threshold (θ_{dep}), 157
 - Observation threshold (θ_{obs}), 157
- Decision mining, 213
 - Inconsistent attributes, 225
 - Unassigned attributes, 225
- Decision tree
 - Builder ($buildTree$), 28
 - Observation instances (OI), 27
- Dependency conditions, 151

- Negative candidate activities, 155
 Observation instances, 155
 Quality, 156
- Event abstraction
 High-level activities, 179
 Low-level activities, 179
 Shared functionality, 181
- Event log, 24
 Activities (Σ), 24
 Attribute function (#), 24
 Latest attribute values (*latest*), 25
 Predecessor ($\bullet e$), 25
 Successor ($e \bullet$), 25
 Traces (\mathcal{E}), 24
- Fitness measure (*fitness*), 65
 Function
 Anonymous partial function (\rightarrow), 20
 Domain (*dom*), 20
 Empty function (\emptyset), 20
 Overriding union (\oplus), 20
 Partial function (\leftrightarrow), 20
 Range (*rng*), 20
 Total function (\rightarrow), 20
- Guided Process Discovery
 Activity patterns (AP), 185
 Alignment steps, 196
 Atomic activity patterns, 185
 Choice composition (\otimes), 190
 Composite activity patterns, 185
 Composition function (AP), 190
 Expansion step, 201
 Expert knowledge, 188
 Global matching error, 199
 GPD, 184
 High-level mapping (*hl*), 185
 Interleaving composition (\leftrightarrow), 190
 Life-cycle transition mapping (*lt*), 185
 Local matching error, 200
 Manual patterns, 188
 Parallel composition (\diamond), 190
- Repetition composition ([n, m]), 191
 Sequence composition (\odot), 190
- Life-cycle transitions (LT), 185
- Mixed Integer Linear Programming
 Incompatible domains, 86
 MILP, 83
 Non-linear constraints, 85
- Multiset, 20
 Difference (\setminus), 21
 Sum (\uplus), 21
- Noise filtering, 147
- Overlapping decision rules, 217
- Perspectives
 Control-flow, 12
 Data, 13
 Function, 13
 Resource, 13
 Time, 13
- Petri net, 32
 Marking (M), 32
 Place (P), 32
 Postset ($p\bullet$), 32
 Preset ($\bullet p$), 32
 Token, 32
 Transition (T), 32
 Transition firing, 33
- Precision
 Activity-precision ($precision_{act}$), 121
 Local behavior, 127
 Local measure, 128
 Measure *precision*, 121
 Observed behavior (*obs*), 120
 Possible behavior (*pos*), 120
 Process state (*state*), 118
 Resource-precision ($precision_{res}$), 125
- Process discovery
 Granularity, 144
 Incompleteness, 143
 Noise, 144

Process model, 29

Relation, 19

Sequence, 21

- Concatenation (\cdot), 21
- Kleene star (X^*), 21
- Prefixes (*prefix*), 22
- Projection (*proj*), 22
- Sum (Σ), 21

Set, 19

- Power set (\mathbb{P}), 19

Trace set (TS), 30

- Labeled trace set, 31

- Process steps (PS), 30

Variables

- Variable assignments (\mathcal{U}_{dom}^X), 22
- Variable domain (*dom*), 22

Variables (V), 22

Summary

Multi-perspective Process Mining

The efficient and effective handling of its processes is essential for the success of an organization. This thesis is about process mining, i. e., analyzing the processes of an organization by using recorded execution data. During the handling of a case (i. e., an instance of a process), data about the execution of activities that are supported by information systems is recorded in databases. We use such process execution data to gain insights about the real execution of processes. Processes can be looked upon from different perspectives. One possible perspective, which has been dominant in research on process mining, is the control-flow perspective. The control-flow perspective is concerned with the possible ordering of activities in the process execution.

In this thesis, we address research challenges in the field of process mining in which a multi-perspective view on processes is needed. We consider problems which require to look beyond the control-flow perspective. In particular, we consider problems in which multiple interacting process perspectives are considered together. We focus on the interaction between the following five perspectives on a process: the control-flow perspective, the resource perspective, which is concerned with the resources (human and non-human) executing the activities; the time perspective, which is concerned with deadlines, bottlenecks and processing time; the data perspective, which is concerned with the manipulation of data objects and their usage to drive decisions; and the function perspective, which is concerned with the description of work of activities and their composition to higher level activities.

We developed and evaluated several novel process mining methods in the area of multi-perspective process mining. The main contributions made are:

- A multi-perspective conformance checking method that balances the importance of multiple perspectives to provide an alignment between recorded event data and a process model. The method provides reliable diagnostics and quality measures with respect to all perspectives of a process model and an event log.
- A multi-perspective process discovery method that uses domain knowledge expressed as multi-perspective activity patterns to abstract sets of low-level activities instances to high-level activity instances (i. e., considers the function perspective). Grouping multiple low-level events to recognizable activity

instances on a higher abstraction level, helps to discover a process model defined over high-level activities, which can be understood better by stakeholders.

- A multi-perspective process discovery method that uses the data perspective (i. e., data attributes recorded in the event log) to distinguish infrequent paths from random noise. The method uses classification techniques to predict under which conditions certain dependencies in the process occur. Data- and control-flow are learned together, i. e., recorded data values are used to improve the discovered control-flow.

All proposed methods have been extensively evaluated in real-life settings by applying them within four case studies that were conducted in the context of processes at three locations: an Italian local police force, a regional hospital in The Netherlands, and a university hospital in Norway. We show that process mining results can be considerably improved upon the state-of-the-art by considering the influence of and the interplay between multiple process perspectives.

Curriculum Vitae

Felix Mannhardt was born on 12/11/1984 in Neuss, Germany. He received a Bachelor in Business Information Systems and a Master in Computer Science from Bonn-Rhein-Sieg University of Applied Sciences in Sankt Augustin, Germany. His Master's thesis was about integrating a workflow management system (YAWL) with a process model repository (Apromore). From 2013 on he started a PhD project at Eindhoven University of Technology in Eindhoven, The Netherlands of which the results are presented in this dissertation. Since August 2017 he is working at the research institute SINTEF in Trondheim, Norway continuing his research on process mining and its application in real-world settings in a new environment.

Felix Mannhardt has received the following awards:

- The *Best Student Paper Award* at the 14th Business Process Management conference in 2016 for the paper *From Low- Level Events to Activities - A Pattern-Based Approach*. Chapter 9 of this thesis is based on this paper.
- The *Best Demo Award* at the 13th Business Process Management conference in 2015 for the *Multi-perspective Process Explorer*, which is presented in Section 11.2.

Felix Mannhardt has the following publications:

Journals

- F. Mannhardt et al. "Guided Process Discovery - A Pattern-based Approach". In: *Inf Syst* (2017). submitted.
- F. Mannhardt et al. "Balanced multi-perspective checking of process conformance". In: *Computing* 98.4 (2016), pp. 407–437. doi: 10.1007/s00607-015-0441-1.

Proceedings and Workshop Contributions

- F. Mannhardt and D. Blinde. "Analyzing the Trajectories of Patients with Sepsis using Process Mining". In: *RADAR+EMISA 2017*. Vol. 1859. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 72–80. url: <http://ceur-ws.org/Vol-1859/bpmds-08-paper.pdf>.

- F. Mannhardt, M. de Leoni, and H. A. Reijers. "Heuristic Mining Revamped: An Interactive Data-aware and Conformance-aware Miner". In: *BPM 2017 Demos*. Vol. 1920. CEUR Workshop Proceedings. CEUR-WS.org, 2017. url: http://ceur-ws.org/Vol-1920/BPM_2017_paper_167.pdf.
- F. Mannhardt and N. Tax. "Unsupervised Event Abstraction using Pattern Abstraction and Local Process Models". In: *RADAR+EMISA 2017*. Vol. 1859. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 55–63. url: <http://ceur-ws.org/Vol-1859/bpmds-06-paper.pdf>.
- F. Mannhardt et al. "Data-driven Process Discovery - Revealing Conditional Infrequent Behavior from Event Logs". In: *CAiSE 2017*. Vol. 10253. LNCS. 2017, pp. 545–560. doi: [10.1007/978-3-319-59536-8_34](https://doi.org/10.1007/978-3-319-59536-8_34).
- F. Mannhardt et al. "Decision Mining Revisited - Discovering Overlapping Rules". In: *CAiSE 2016*. Vol. 9694. Lecture Notes in Computer Science. Springer, 2016, pp. 377–392. doi: [10.1007/978-3-319-39696-5_23](https://doi.org/10.1007/978-3-319-39696-5_23).
- F. Mannhardt et al. "From Low-Level Events to Activities - A Pattern-Based Approach". In: *BPM 2016*. Vol. 9850. Lecture Notes in Computer Science. Springer, 2016, pp. 125–141. doi: [10.1007/978-3-319-45348-4_8](https://doi.org/10.1007/978-3-319-45348-4_8).
- H. van der Aa et al. "On the Fragmentation of Process Information: Challenges, Solutions, and Outlook". In: *BPMDS 2015*. Vol. 214. LNBIP. Springer, 2015, pp. 3–18. doi: [10.1007/978-3-319-19237-6_1](https://doi.org/10.1007/978-3-319-19237-6_1).
- F. Mannhardt, M. de Leoni, and H. A. Reijers. "The Multi-perspective Process Explorer". In: *BPM 2015 Demos*. Vol. 1418. CEUR Workshop Proceedings. CEUR-WS.org, 2015, pp. 130–134. url: <http://ceur-ws.org/Vol-1418/paper27.pdf>.
- F. Mannhardt et al. "Measuring the Precision of Multi-perspective Process Models". In: *BPM 2015 Workshops*. Vol. 256. LNBIP. Springer, 2015, pp. 113–125. doi: [10.1007/978-3-319-42887-1_10](https://doi.org/10.1007/978-3-319-42887-1_10).
- F. Mannhardt, M. de Leoni, and H. A. Reijers. "Extending Process Logs with Events from Supplementary Sources". In: *BPM 2014 Workshops*. Vol. 202. LNBIP. Springer, 2014, pp. 235–247. doi: [10.1007/978-3-319-15895-2_21](https://doi.org/10.1007/978-3-319-15895-2_21).
- F. Mannhardt. "Web-based Editor for YAWL". In: *YAWL Symposium 2013*. Vol. 982. CEUR Workshop Proceedings. CEUR-WS.org, 2013, pp. 62–68. url: <http://ceur-ws.org/Vol-982/YAWL2013-Paper09.pdf>.
- C. C. Ekanayake et al. "Detecting Approximate Clones in Process Model Repositories with Apromore". In: *BPM 2012 Demos*. Vol. 940. CEUR Workshop Proceedings. CEUR-WS.org, 2012, pp. 29–33. url: <http://ceur-ws.org/Vol-940/paper6.pdf>.

Technical Reports (Non-Refereed)

- F. Mannhardt. *XESLite - Managing Large XES Event Logs in ProM*. BPM Center Report BPM-16-04. BPMCenter.org, 2016. URL: <http://bpmcenter.org/wp-content/uploads/reports/2016/BPM-16-04.pdf>.
- F. Mannhardt et al. *Decision Mining Revisited - Discovering Overlapping Rules*. BPM Center Report BPM-16-01. BPMcenter.org, 2016. URL: <http://bpmcenter.org/wp-content/uploads/reports/2016/BPM-16-01.pdf>.
- F. Mannhardt et al. *From Low-Level Events to Activities - A Pattern-based Approach*. BPM Center Report BPM-16-02. BPMCenter.org, 2016. URL: <http://bpmcenter.org/wp-content/uploads/reports/2016/BPM-16-02.pdf>.
- H. Verbeek and F. Mannhardt. *The DrFurby Classifier submission to the Process Discovery Contest @ BPM 2016*. BPM Center Report BPM-16-08. BPMCenter.org, 2016. URL: <http://bpmcenter.org/wp-content/uploads/reports/2016/BPM-16-08.pdf>.
- F. Mannhardt et al. *Balanced Multi-Perspective Checking of Process Conformance*. BPM Center Report BPM-14-07. BPMcenter.org, 2014. URL: <http://bpmcenter.org/wp-content/uploads/reports/2014/BPM-14-07.pdf>.

Data sets

- F. Mannhardt. *Hospital Billing - Event Log*. Eindhoven University of Technology. Dataset. 2017. doi: [10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfb741](https://doi.org/10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfb741).
- F. Mannhardt. *Data-driven Process Discovery - Artificial Event Log*. Dataset. 2016. doi: [10.4121/uuid:32cad43f-8bb9-46af-8333-48aae2bea037](https://doi.org/10.4121/uuid:32cad43f-8bb9-46af-8333-48aae2bea037).
- F. Mannhardt. *Sepsis Cases - Event Log*. Dataset. 2016. doi: [10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460](https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460).
- M. de Leoni and F. Mannhardt. *Road Traffic Fine Management Process*. Dataset. 2015. doi: [10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5](https://doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5).

Acknowledgments

Looking back on the last years, I was first exposed to the idea of pursuing a PhD in sunny Brisbane, Australia. I wrote my Master's thesis during a visit at the BPM research group of the QUT. I owe a lot of gratitude to my thesis adviser *Andreas Hense* and to my supervisors at QUT, *Marcello* and *Arthur*, who encouraged me to start PhD studies. Whereas, I truly enjoyed my time at QUT, my PhD journey took me elsewhere. Brisbane is just located too far away from home. I was lucky to meet *Hajo* during one of our regular lunch walks in the Brisbane city center. Entirely unaware of how my future would unfold, I cannot think of a better way of having met him. Fast forwarding a little bit more than half a year in the future (April 2013), I was starting my PhD studies with *Hajo* and *Wil* in Eindhoven.

First, I want to thank *Hajo* for the opportunity to do my PhD studies with him and for the excellent guidance in the last years. It was a pleasure to work with (*under* would be the wrong word) *Hajo* both personally and professionally. I think you cannot wish for a better PhD adviser. Due to several external factors the journey was not always smooth. We both started as being funded by Perceptive Software, but already after half of the time the funding was cut. An experience I would rather not like to repeat. I had luck in the misfortune and was given the funds to finish my PhD (many thanks to the ones responsible for this). *Hajo* started his research group in Amsterdam. Thanks for his continued full support for my PhD project in Eindhoven.

Wil, I also met the first time in Brisbane. I was very glad to have him attending my Master's thesis defense. Of course, he asked the one difficult question. In the last four years he was always a source of inspiration and asked plenty of more questions that challenged me to grow a lot. Having both *Hajo* and *Wil* as PhD advisers was definitely a lucky draw. Their feedback often complemented each other's very nicely, which is a luxury not all PhD students get to experience.

My advisory team would not have been complete without the daily help of *Massimiliano*. Not only offered he guidance and support with an always open office door but also friendship beyond the PhD work. I enjoyed all the shared conference visits, lunch breaks, BBQs, etc. Often, he would spend extra time to help shaping my research. I hope the fact that I often needed more than one discussion to be convinced of something did not cost you too much patience.

Special thanks to *Ine* and *Riet* for helping with any administrative problems that can be encountered. Further thanks to *Ine* for reducing the number of spelling mistakes and for help with the practicalities of this dissertation. I would like to also thank all my external PhD committee members *Jan Vanthienen*, *Pieter Toussaint*,

Remco Dijkman, and *Josep Carmona*. Thank you for your time reading the many pages (sorry!) of my dissertation and the helpful comments. Special thanks to Pieter for initiating the contact to SINTEF in Trondheim to help with the continuation of my journey. I would also like to thank all the people that helped with the case studies presented in this thesis. Thanks Daan, Janet, Erick, Patrick, and Ivar for helping me to get and understand the data.

What would PhD studies be without colleagues, visitors, and fellow PhD students, who share the joy and (maybe that divides) the pain of doing a PhD. One joy of doing a PhD (at least in a large group as in Eindhoven) is that you meet many people. I would like to thank all of them for the great time we spent together and I hope to see many of you again at some stage in my life. In pseudo-random order: Joos, Jorge, Dennis, Sander (definitely the office with the best music), Rafael, Elham, Arya, Boudewijn, Eric, Dirk, Natalia, Mykola, George, Paul, Christian, Michael, Xixi, Yulia, Han, Maikel van E., Murat, Marcus, Bas, Eduardo, Alfredo, Alok, Niek, Guangming, Shiva, Rémi, Bart, Maikel L., Cong, Mohammadreza, Nour, Alifah, Farideh, Renata, Marwan, and all the people I met at conference or visits abroad.

The last people to acknowledge are surely the most important. Many thanks to my family for supporting me with all my goals and my (almost) endless study (something my grandfather would probably never have understood). I owe the biggest thank of them all (even that superlative does not suffice) to my love and wife: *Danni*. Without your love, unconditional support, and understanding, I could not have finished this thesis. Thanks for the time you sacrificed for this. I promise, there won't be a next PhD thesis and that there is more time for the youngest contributor to this thesis: *Joris*.

Felix Mannhardt
Trondheim, December 2017

SIKS Dissertations

2011

- 01 Botond Cseke (RUN), *Variational Algorithms for Bayesian Inference in Latent Gaussian Models.*
- 02 Nick Tinnemeier(UU), *Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language.*
- 03 Jan Martijn van der Werf (TUE), *Compositional Design and Verification of Component-Based Information Systems.*
- 04 Hado van Hasselt (UU), *Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning.*
- 05 Base van der Raadt (VU), *Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline..*
- 06 Yiwen Wang (TUE), *Semantically-Enhanced Recommendations in Cultural Heritage.*
- 07 Yujia Cao (UT), *Multimodal Information Presentation for High Load Human Computer Interaction.*
- 08 Nieske Vergunst (UU), *BDI-based Generation of Robust Task-Oriented Dialogues.*
- 09 Tim de Jong (OU), *Contextualised Mobile Media for Learning.*
- 10 Bart Bogaert (UvT), *Cloud Content Contention.*
- 11 Dhaval Vyas (UT), *Designing for Awareness: An Experience-focused HCI Perspective.*
- 12 Carmen Bratosin (TUE), *Grid Architecture for Distributed Process Mining.*
- 13 Xiaoyu Mao (UvT), *Airport under Control. Multiagent Scheduling for Airport Ground Handling.*
- 14 Milan Lovric (EUR), *Behavioral Finance and Agent-Based Artificial Markets.*
- 15 Marijn Koolen (UvA), *The Meaning of Structure: the Value of Link Evidence for Information Retrieval.*
- 16 Maarten Schadd (UM), *Selective Search in Games of Different Complexity.*
- 17 Jiyin He (UVA), *Exploring Topic Structure: Coherence, Diversity and Relatedness.*
- 18 Mark Ponsen (UM), *Strategic Decision-Making in complex games.*
- 19 Ellen Rusman (OU), *The Mind ' s Eye on Personal Profiles.*
- 20 Qing Gu (VU), *Guiding service-oriented software engineering - A view-based approach.*
- 21 Linda Terlouw (TUD), *Modularization and Specification of Service-Oriented Systems.*
- 22 Junte Zhang (UVA), *System Evaluation of Archival Description and Access.*
- 23 Wouter Weerkamp (UVA), *Finding People and their Utterances in Social Media.*
- 24 Herwin van Welbergen (UT), *Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior.*
- 25 Syed Waqar ul Qounain Jaffry (VU)), *Analysis and Validation of Models for Trust Dynamics.*
- 26 Matthijs Aart Pontier (VU), *Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots.*
- 27 Aniel Bhulai (VU), *Dynamic website optimization through autonomous management of design patterns.*
- 28 Rianne Kaptein(UVA), *Effective Focused Retrieval by Exploiting Query Context and Document Structure.*
- 29 Faisal Kamiran (TUE), *Discrimination-aware Classification.*
- 30 Egon van den Broek (UT), *Affective Signal Processing (ASP): Unraveling the mystery of emotions.*
- 31 Ludo Waltman (EUR), *Computational and Game-Theoretic Approaches for Modeling Bounded Rationality.*
- 32 Nees-Jan van Eck (EUR), *Methodological Advances in Bibliometric Mapping of Science.*
- 33 Tom van der Weide (UU), *Arguing to Motivate Decisions.*
- 34 Paolo Turrini (UU), *Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations.*
- 35 Maaike Harbers (UU), *Explaining Agent Behavior in Virtual Environments.*

- tual Training.*
- 36** Erik van der Spek (UU), *Experiments in serious game design: a cognitive approach.*
- 37** Adriana Burlutiu (RUN), *Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference.*
- 38** Nyree Lemmens (UM), *Bee-inspired Distributed Optimization.*
- 39** Joost Westra (UU), *Organizing Adaptation using Agents in Serious Games.*
- 40** Viktor Clerc (VU), *Architectural Knowledge Management in Global Software Development.*
- 41** Luan Ibraimi (UT), *Cryptographically Enforced Distributed Data Access Control.*
- 42** Michal Sindlar (UU), *Explaining Behavior through Mental State Attribution.*
- 43** Henk van der Schuur (UU), *Process Improvement through Software Operation Knowledge.*
- 44** Boris Reuderink (UT), *Robust Brain-Computer Interfaces.*
- 45** Herman Stehouwer (UvT), *Statistical Language Models for Alternative Sequence Selection.*
- 46** Beibei Hu (TUD), *Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work.*
- 47** Azizi Bin Ab Aziz(VU), *Exploring Computational Models for Intelligent Support of Persons with Depression.*
- 48** Mark Ter Maat (UT), *Response Selection and Turn-taking for a Sensitive Artificial Listening Agent.*
- 49** Andreea Niculescu (UT), *Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality.*
- 2012**
- 01** Terry Kakeeto (UvT), *Relationship Marketing for SMEs in Uganda.*
- 02** Muhammad Umair(VU), *Adaptivity, emotion, and Rationality in Human and Ambient Agent Models.*
- 03** Adam Vanya (VU), *Supporting Architecture Evolution by Mining Software Repositories.*
- 04** Jurriaan Souer (UU), *Development of Content Management System-based Web Applications.*
- 05** Marijn Plomp (UU), *Maturing Interorganisational Information Systems.*
- 06** Wolfgang Reinhardt (OU), *Awareness Support for Knowledge Workers in Research Networks.*
- 07** Rianne van Lambalgen (VU), *When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions.*
- 08** Gerben de Vries (UVA), *Kernel Methods for Vessel Trajectories.*
- 09** Ricardo Neisse (UT), *Trust and Privacy Management Support for Context-Aware Service Platforms.*
- 10** David Smits (TUE), *Towards a Generic Distributed Adaptive Hypermedia Environment.*
- 11** J.C.B. Rantham Prabhakara (TUE), *Process Mining in the Large: Preprocessing, Discovery, and Diagnostics.*
- 12** Kees van der Sluijs (TUE), *Model Driven Design and Data Integration in Semantic Web Information Systems.*
- 13** Suleman Shahid (UvT), *Fun and Face: Exploring non-verbal expressions of emotion during playful interactions.*
- 14** Evgeny Knutov(TUE), *Generic Adaptation Framework for Unifying Adaptive Web-based Systems.*
- 15** Natalie van der Wal (VU), *Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes..*
- 16** Fiemke Both (VU), *Helping people by understanding them - Ambient Agents supporting task execution and depression treatment.*
- 17** Amal Elgammal (UvT), *Towards a Comprehensive Framework for Business Process Compliance.*
- 18** Eltjo Poort (VU), *Improving Solution Architecting Practices.*
- 19** Helen Schonenberg (TUE), *What's Next? Operational Support for Business Process Execution.*
- 20** Ali Bahramisharif (RUN), *Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing.*
- 21** Roberto Cornacchia (TUD), *Querying Sparse Matrices for Information Retrieval.*
- 22** Thijs Vis (UvT), *Intelligence, politie en veiligheidsdienst: verenigbare grootheden?.*
- 23** Christian Muehl (UT), *Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction.*
- 24** Laurens van der Werff (UT), *Evaluation of Noisy Transcripts for Spoken Document Retrieval.*

- 25** Silja Eckartz (UT), *Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application.*
- 26** Emile de Maat (UVA), *Making Sense of Legal Text.*
- 27** Hayrettin Gurkok (UT), *Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games.*
- 28** Nancy Pascall (UvT), *Engendering Technology Empowering Women.*
- 29** Almer Tigelaar (UT), *Peer-to-Peer Information Retrieval.*
- 30** Alina Pommeranz (TUD), *Designing Human-Centered Systems for Reflective Decision Making.*
- 31** Emily Bagarukayo (RUN), *A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure.*
- 32** Wietske Visser (TUD), *Qualitative multi-criteria preference representation and reasoning.*
- 33** Rory Sie (OUN), *Coalitions in Cooperation Networks (CO-COON).*
- 34** Pavol Jancura (RUN), *Evolutionary analysis in PPI networks and applications.*
- 35** Evert Haasdijk (VU), *Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics.*
- 36** Denis Ssebugawo (RUN), *Analysis and Evaluation of Collaborative Modeling Processes.*
- 37** Agnes Nakakawa (RUN), *A Collaboration Process for Enterprise Architecture Creation.*
- 38** Selmar Smit (VU), *Parameter Tuning and Scientific Testing in Evolutionary Algorithms.*
- 39** Hassan Fatemi (UT), *Risk-aware design of value and coordination networks.*
- 40** Agus Gunawan (UvT), *Information Access for SMEs in Indonesia.*
- 41** Sebastian Kelle (OU), *Game Design Patterns for Learning.*
- 42** Dominique Verpoorten (OU), *Reflection Amplifiers in self-regulated Learning.*
- 43** Withdrawn, .
- 44** Anna Tordai (VU), *On Combining Alignment Techniques.*
- 45** Benedikt Kratz (UvT), *A Model and Language for Business-aware Transactions.*
- 46** Simon Carter (UVA), *Exploration and Exploitation of Multilingual Data for Statistical Machine Translation.*
- 47** Manos Tsagkias (UVA), *Mining Social Media: Tracking Content and Predicting Behavior.*
- 48** Jorn Bakker (TUE), *Handling Abrupt Changes in Evolving Time-series Data.*
- 49** Michael Kaisers (UM), *Learning against Learning – Evolutionary dynamics of reinforcement learning algorithms in strategic interactions.*
- 50** Steven van Kerrel (TUD), *Ontology driven Enterprise Information Systems Engineering.*
- 51** Jeroen de Jong (TUD), *Heuristics in Dynamic Scheduling; a practical framework with a case study in elevator dispatching.*
- 2013**
- 01** Viorel Milea (EUR), *News Analytics for Financial Decision Support.*
- 02** Erietta Liarou (CWI), *MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing.*
- 03** Szymon Klarman (VU), *Reasoning with Contexts in Description Logics.*
- 04** Chetan Yadati(TUD), *Coordinating autonomous planning and scheduling.*
- 05** Dulce Pumareja (UT), *Groupware Requirements Evolutions Patterns.*
- 06** Romulo Goncalves(CWI), *The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience.*
- 07** Giel van Lankveld (UvT), *Quantifying Individual Player Differences.*
- 08** Robbert-Jan Merk(VU), *Making enemies: cognitive modeling for opponent agents in fighter pilot simulators.*
- 09** Fabio Gori (RUN), *Metagenomic Data Analysis: Computational Methods and Applications.*
- 10** Jeewanie Jayasinghe Arachchige(UvT), *A Unified Modeling Framework for Service Design..*
- 11** Evangelos Pournaras(TUD), *Multi-level Reconfigurable Self-organization in Overlay Services.*
- 12** Marian Razavian(VU), *Knowledge-driven Migration to Services.*
- 13** Mohammad Safiri(UT), *Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly.*
- 14** Jafar Tanha (UVA), *Ensemble Approaches to Semi-Supervised Learning Learning.*

- 15** Daniel Hennes (UM), *Multiagent Learning - Dynamic Games and Applications*.
- 16** Eric Kok (UU), *Exploring the practical benefits of argumentation in multi-agent deliberation*.
- 17** Koen Kok (VU), *The PowerMatcher: Smart Coordination for the Smart Electricity Grid*.
- 18** Jeroen Janssens (UvT), *Outlier Selection and One-Class Classification*.
- 19** Renze Steenhuizen (TUD), *Coordinated Multi-Agent Planning and Scheduling*.
- 20** Katja Hofmann (UvA), *Fast and Reliable Online Learning to Rank for Information Retrieval*.
- 21** Sander Wubben (UvT), *Text-to-text generation by monolingual machine translation*.
- 22** Tom Claassen (RUN), *Causal Discovery and Logic*.
- 23** Patrício de Alencar Silva(UvT), *Value Activity Monitoring*.
- 24** Haitham Bou Ammar (UM), *Automated Transfer in Reinforcement Learning*.
- 25** Agnieszka Anna Latoszek-Berendsen (UM), *Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System*.
- 26** Alireza Zarghami (UT), *Architectural Support for Dynamic Homecare Service Provisioning*.
- 27** Mohammad Huq (UT), *Inference-based Framework Managing Data Provenance*.
- 28** Frans van der Sluis (UT), *When Complexity becomes Interesting: An Inquiry into the Information eXperience*.
- 29** Iwan de Kok (UT), *Listening Heads*.
- 30** Joyce Nakatumba (TUE), *Resource-Aware Business Process Management: Analysis and Support*.
- 31** Dinh Khoa Nguyen (UvT), *Blueprint Model and Language for Engineering Cloud Applications*.
- 32** Kamakshi Rajagopal (OUN), *Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development*.
- 33** Qi Gao (TUD), *User Modeling and Personalization in the Microblogging Sphere*.
- 34** Kien Tjin-Kam-Jet (UT), *Distributed Deep Web Search*.
- 35** Abdallah El Ali (UvA), *Minimal Mobile Human Computer Interaction*.
- 36** Than Lam Hoang (TUE), *Pattern Mining in Data Streams*.
- 37** Dirk Börner (OUN), *Ambient Learning Displays*.
- 38** Eelco den Heijer (VU), *Autonomous Evolutionary Art*.
- 39** Joop de Jong (TUD), *A Method for Enterprise Ontology based Design of Enterprise Information Systems*.
- 40** Pim Nijssen (UM), *Monte-Carlo Tree Search for Multi-Player Games*.
- 41** Jochem Liem (UVA), *Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning*.
- 42** Léon Planken (TUD), *Algorithms for Simple Temporal Reasoning*.
- 43** Marc Bron (UVA), *Exploration and Contextualization through Interaction and Concepts*.
- 2014**
- 01** Nicola Barile (UU), *Studies in Learning Monotone Models from Data*.
- 02** Fiona Tulyano (RUN), *Combining System Dynamics with a Domain Modeling Method*.
- 03** Sergio Raul Duarte Torres (UT), *Information Retrieval for Children: Search Behavior and Solutions*.
- 04** Hanna Jochmann-Mannak (UT), *Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation*.
- 05** Jurriaan van Reijsen (UU), *Knowledge Perspectives on Advancing Dynamic Capability*.
- 06** Damian Tamburri (VU), *Supporting Networked Software Development*.
- 07** Arya Adriansyah (TUE), *Aligning Observed and Modeled Behavior*.
- 08** Samur Araujo (TUD), *Data Integration over Distributed and Heterogeneous Data Endpoints*.
- 09** Philip Jackson (UvT), *Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language*.
- 10** Ivan Salvador Razo Zapata (VU), *Service Value Networks*.
- 11** Janneke van der Zwaan (TUD), *An Empathic Virtual Buddy for Social Support*.
- 12** Willem van Willigen (VU), *Look Ma, No Hands: Aspects of Autonomous Vehicle Control*.
- 13** Arlette van Wissen (VU), *Agent-Based Support for Behavior Change: Models and Applications in Health and Safety*

- Domains.*
- 14** Yangyang Shi (TUD), *Language Models With Meta-information*
- 15** Natalya Mogle (VU), *Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare.*
- 16** Krystyna Milian (VU), *Supporting trial recruitment and design by automatically interpreting eligibility criteria.*
- 17** Kathrin Dentler (VU), *Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability.*
- 18** Mattij Ghijsen (UVA), *Methods and Models for the Design and Study of Dynamic Agent Organizations.*
- 19** Vinicius Ramos (TUE), *Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support.*
- 20** Mena Habib (UT), *Named Entity Extraction and Disambiguation for Informal Text: The Missing Link.*
- 21** Kassidy Clark (TUD), *Negotiation and Monitoring in Open Environments.*
- 22** Marieke Peeters (UU), *Personalized Educational Games - Developing agent-supported scenario-based training.*
- 23** Eleftherios Sidiropoulos (UvA/CWI), *Space Efficient Indexes for the Big Data Era.*
- 24** Davide Ceolin (VU), *Trusting Semi-structured Web Data.*
- 25** Martijn Lappenschaar (RUN), *New network models for the analysis of disease interaction.*
- 26** Tim Baarslag (TUD), *What to Bid and When to Stop.*
- 27** Rui Jorge Almeida (EUR), *Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty.*
- 28** Anna Chmielowiec (VU), *Decentralized k-Clique Matching.*
- 29** Jaap Kabbedijk (UU), *Variability in Multi-Tenant Enterprise Software.*
- 30** Peter de Cock (UvT), *Anticipating Criminal Behaviour.*
- 31** Leo van Moergestel (UU), *Agent Technology in Agile Multiparallel Manufacturing and Product Support.*
- 32** Naser Ayat (UvA), *On Entity Resolution in Probabilistic Data.*
- 33** Tesfa Tegegne (RUN), *Service Discovery in eHealth.*
- 34** Christina Manteli(VU), *The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems..*
- 35** Joost van Ooijen (UU), *Cognitive Agents in Virtual Worlds:*
- A Middleware Design Approach.*
- 36** Joos Buijs (TUE), *Flexible Evolutionary Algorithms for Mining Structured Process Models.*
- 37** Maral Dadvar (UT), *Experts and Machines United Against Cyberbullying.*
- 38** Danny Plass-Oude Bos (UT), *Making brain-computer interfaces better: improving usability through post-processing..*
- 39** Jasmina Maric (UvT), *Web Communities, Immigration, and Social Capital.*
- 40** Walter Omona (RUN), *A Framework for Knowledge Management Using ICT in Higher Education.*
- 41** Frederic Hogenboom (EUR), *Automated Detection of Financial Events in News Text.*
- 42** Carsten Eijckhof (CWI/TUD), *Contextual Multidimensional Relevance Models.*
- 43** Kevin Vlaanderen (UU), *Supporting Process Improvement using Method Increments.*
- 44** Paulien Meesters (UvT), *Intelligent Blauw. Met als ondertitel: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden..*
- 45** Birgit Schmitz (OUN), *Mobile Games for Learning: A Pattern-Based Approach.*
- 46** Ke Tao (TUD), *Social Web Data Analytics: Relevance, Redundancy, Diversity.*
- 47** Shangsong Liang (UVA), *Fusion and Diversification in Information Retrieval.*
- 2015**
- 01** Niels Netten (UvA), *Machine Learning for Relevance of Information in Crisis Response.*
- 02** Faiza Bukhsh (UvT), *Smart auditing: Innovative Compliance Checking in Customs Controls.*
- 03** Twan van Laarhoven (RUN), *Machine learning for network data.*
- 04** Howard Spoelstra (OUN), *Collaborations in Open Learning Environments.*
- 05** Christoph Bösch(UT), *Cryptographically Enforced Search Pattern Hiding.*
- 06** Farideh Heidari (TUD), *Business Process Quality Computation - Computing Non-Functional Requirements to Improve Business Processes.*
- 07** Maria-Hendrike Peetz(UvA), *Time-Aware Online Reputation Management.*

- tation Analysis.*
- 08** Jie Jiang (TUD), *Organizational Compliance: An agent-based model for designing and evaluating organizational interactions.*
- 09** Randy Klaassen(UT), *HCI Perspectives on Behavior Change Support Systems.*
- 10** Henry Hermans (OUN), *OpenUI: design of an integrated system to support lifelong learning.*
- 11** Yongming Luo(TUE), *Designing algorithms for big graph datasets: A study of computing bisimulation and joins.*
- 12** Julie M. Birkholz (VU), *Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks.*
- 13** Giuseppe Procaccianti(VU), *Energy-Efficient Software.*
- 14** Bart van Straalen (UT), *A cognitive approach to modeling bad news conversations.*
- 15** Klaas Andries de Graaf (VU), *Ontology-based Software Architecture Documentation.*
- 16** Changyun Wei (UT), *Cognitive Coordination for Cooperative Multi-Robot Teamwork.*
- 17** André van Cleeff (UT), *Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs.*
- 18** Holger Pirk (CWI), *Waste Not, Want Not! - Managing Relational Data in Asymmetric Memories.*
- 19** Bernardo Tabuena (OUN), *Ubiquitous Technology for Lifelong Learners.*
- 20** Loïs Vanhée(UU), *Using Culture and Values to Support Flexible Coordination.*
- 21** Sibren Fetter (OUN), *Using Peer-Support to Expand and Stabilize Online Learning.*
- 23** Luit Gazendam (VU), *Cataloguer Support in Cultural Heritage.*
- 24** Richard Berendsen (UVA), *Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation.*
- 25** Steven Woudenberg (UU), *Bayesian Tools for Early Disease Detection.*
- 26** Alexander Hogenboom (EUR), *Sentiment Analysis of Text Guided by Semantics and Structure.*
- 27** Sándor Héman (CWI), *Updating compressed column stores.*
- 28** Janet Bagorogoza(TiU), *KNOWLEDGE MANAGEMENT AND HIGH PERFORMANCE; The Uganda Financial Institutions Model for HPO.*
- 29** Hendrik Baier (UM), *Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains.*
- 30** Kiavash Bahreini(OU), *Real-time Multimodal Emotion Recognition in E-Learning.*
- 31** Yakup Koç (TUD), *On the robustness of Power Grids.*
- 32** Jerome Gard(UL), *Corporate Venture Management in SMEs.*
- 33** Frederik Schadd (TUD), *Ontology Mapping with Auxiliary Resources.*
- 34** Victor de Graaf(UT), *Gesocial Recommender Systems.*
- 35** Jungxiao Xu (TUD), *Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction.*
- ## 2016
- 01** Syed Saider Abbas (RUN), *Recognition of Shapes by Humans and Machines.*
- 02** Michiel Christiaan Meulendijk (UU), *Optimizing medication reviews through decision support: prescribing a better pill to swallow.*
- 03** Maya Sappelli (RUN), *Knowledge Work in Context: User Centered Knowledge Worker Support.*
- 04** Laurens Rietveld (VU), *Publishing and Consuming Linked Data.*
- 05** Evgeny Sherkhonov (UVA), *Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers.*
- 06** Michel Wilson (TUD), *Robust scheduling in an uncertain environment.*
- 07** Jeroen de Man (VU), *Measuring and modeling negative emotions for virtual training.*
- 08** Matje van de Camp (TiU), *A Link to the Past: Constructing Historical Social Networks from Unstructured Data.*
- 09** Archana Nottamkandath (VU), *Trusting Crowdsourced Information on Cultural Artefacts.*
- 10** George Karafotias (VUA), *Parameter Control for Evolutionary Algorithms.*
- 11** Anne Schuth (UVA), *Search Engines that Learn from Their Users.*
- 12** Max Knobbe(UU), *Logics for Modelling and Verifying Normative Multi-Agent Systems.*
- 13** Nana Baah Gyan (VU), *The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach.*
- 14** Ravi Khadka (UU), *Revisiting Legacy Software System Modernization.*

- 15** Steffen Michels (RUN), *Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments.*
- 16** Guangliang Li (UVA), *Socially Intelligent Autonomous Agents that Learn from Human Reward.*
- 17** Berend Weel (VU), *Towards Embodied Evolution of Robot Organisms.*
- 18** Albert Meroño Peñuela (VU), *Refining Statistical Data on the Web.*
- 19** Julia Efremova (Tu/e), *Mining Social Structures from Genealogical Data.*
- 20** Daan Odijk (UVA), *Context & Semantics in News & Web Search.*
- 21** Alejandro Moreno Céller (UT), *From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground.*
- 22** Grace Lewis (VU), *Software Architecture Strategies for Cyber-Foraging Systems.*
- 23** Fei Cai (UVA), *Query Auto Completion in Information Retrieval.*
- 24** Brend Wanders (UT), *Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach.*
- 25** Julia Kiseleva (TU/e), *Using Contextual Information to Understand Searching and Browsing Behavior.*
- 26** Dilhan Thilakarathne (VU), *In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains.*
- 27** Wen Li (TUD), *Understanding Geo-spatial Information on Social Media.*
- 28** Mingxin Zhang (TUD), *Large-scale Agent-based Social Simulation - A study on epidemic prediction and control.*
- 29** Nicolas Höning (TUD), *Peak reduction in decentralised electricity systems -Markets and prices for flexible planning.*
- 30** Ruud Mattheij (UvT), *The Eyes Have It.*
- 31** Mohammad Khelghati (UT), *Deep web content monitoring.*
- 32** Eelco Vriezekolk (UT), *Assessing Telecommunication Service Availability Risks for Crisis Organisations.*
- 33** Peter Bloem (UVA), *Single Sample Statistics, exercises in learning from just one example.*
- 34** Dennis Schunselaar (TUE), *Configurable Process Trees: Elicitation, Analysis, and Enactment.*
- 35** Zhaochun Ren (UVA), *Monitoring Social Media: Summarization, Classification and Recommendation.*
- 36** Daphne Karreman (UT), *Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies.*
- 37** Giovanni Sileno (UvA), *Aligning Law and Action - a conceptual and computational inquiry.*
- 38** Andrea Minuto (UT), *MATERIALS THAT MATTER - Smart Materials meet Art & Interaction Design.*
- 39** Merijn Bruijnes (UT), *Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect.*
- 40** Christian Detweiler (TUD), *Accounting for Values in Design.*
- 41** Thomas King (TUD), *Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance.*
- 42** Spyros Martzoukos (UVA), *Combinatorial and Compositional Aspects of Bilingual Aligned Corpora.*
- 43** Saskia Koldijk (RUN), *Context-Aware Support for Stress Self-Management: From Theory to Practice.*
- 44** Thibault Sellam (UVA), *Automatic Assistants for Database Exploration.*
- 45** Bram van de Laar (UT), *Experiencing Brain-Computer Interface Control.*
- 46** Jorge Gallego Perez (UT), *Robots to Make you Happy.*
- 47** Christina Weber (UL), *Real-time foresight - Preparedness for dynamic innovation networks.*
- 48** Tanja Buttler (TUD), *Collecting Lessons Learned.*
- 49** Gleb Polevoy (TUD), *Participation and Interaction in Projects. A Game-Theoretic Analysis.*
- 50** Yan Wang (UVT), *The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains.*

2017

- 01** Jan-Jaap Oerlemans (UL), *Investigating Cybercrime.*
- 02** Sjoerd Timmer (UU), *Designing and Understanding Forensic Bayesian Networks using Argumentation.*
- 03** Daniël Harold Telgen (UU), *Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines.*
- 04** Mrunal Gawade (CWI), *MULTI-CORE PARALLELISM*

- IN A COLUMN-STORE.**
- 05** Mahdieh Shadi (UVA), *Collaboration Behavior*.
- 06** Damir Vandic (EUR), *Intelligent Information Systems for Web Product Search*.
- 07** Roel Bertens (UU), *Insight in Information: from Abstract to Anomaly*.
- 08** Rob Konijn (VU), *Detecting Interesting Differences: Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery*.
- 09** Dong Nguyen (UT), *Text as Social and Cultural Data: A Computational Perspective on Variation in Text*.
- 10** Robby van Delden (UT), *(Steering) Interactive Play Behavior*.
- 11** Florian Kunneman (RUN), *Modelling patterns of time and emotion in Twitter #anticipointment*.
- 12** Sander Leemans (TUE), *Robust Process Mining with Guarantees*.
- 13** Gijs Huisman (UT), *Social Touch Technology - Extending the reach of social touch through haptic technology*.
- 14** Shoshannah Tekofsky (UvT), *You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior*.
- 15** Peter Berck, Radboud University (RUN), *Memory-Based Text Correction*.
- 16** Aleksandr Chuklin (UVA), *Understanding and Modeling Users of Modern Search Engines*.
- 17** Daniel Dimov (UL), *Crowdsourced Online Dispute Resolution*.
- 18** Ridho Reinanda (UVA), *Entity Associations for Search*.
- 19** Jeroen Vuurens (TUD), *Proximity of Terms, Texts and Semantic Vectors in Information Retrieval*.
- 20** Mohammadbashir Sedighi (TUD), *Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility*.
- 21** Jeroen Linssen (UT), *Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)*.
- 22** Sara Magliacane (VU), *Logics for causal inference under uncertainty*.
- 23** David Graus (UVA), *Entities of Interest—Discovery in Digital Traces*.
- 24** Chang Wang (TUD), *Use of Affordances for Efficient Robot Learning*.
- 25** Veruska Zamborlini (VU), *Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search*.
- 26** Merel Jung (UT), *Socially intelligent robots that understand and respond to human touch*.
- 27** Michiel Joosse (UT), *Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors*.
- 28** John Klein (VU), *Architecture Practices for Complex Contexts*.
- 29** Adel Alhuraibi (UVT), *From IT-Business Strategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT*.
- 30** Wilma Latuny (UVT), *The Power of Facial Expressions*.
- 31** Ben Ruijl (UL), *Advances in computational methods for QFT calculations*.
- 32** Thaer Samar (RUN), *Access to and Retrievability of Content in Web Archives*.
- 33** Brigit van Loggem (OU), *Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity*.
- 34** Maren Scheffel (OUN), *The Evaluation Framework for Learning Analytics*.
- 35** Martine de Vos (VU), *Interpreting natural science spreadsheets*.
- 36** Yuanhao Guo (UL), *Shape Analysis for Phenotype Characterisation from High-throughput Imaging*.
- 37** Alejandro Montes García (TUE), *WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy*.
- 38** Alex Kayal (TUD), *Normative Social Applications*.
- 39** Sara Ahmadi (RUN), *Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR*.
- 40** Altaf Hussain Abro (VUA), *Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems*.
- 41** Adnan Manzoor (VUA), *Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle*.
- 42** Elena Sokolova (RUN), *Causal discovery from mixed and missing data with applications on ADHD datasets*.
- 43** Maaike de Boer (RUN), *Semantic Mapping in Video Retrieval*.
- 44** Garm Lucassen (UU), *Understanding User Stories - Com-*

putational Linguistics in Agile Requirements Engineering.

45 Bas Testerink (UU), *Decentralized Runtime Norm Enforcement.*

46 Jan Schneider (OU), *Sensor-based Learning Support.*

47 Yie Yang (TUD), *Crowd Knowledge Creation Acceleration.*

48 Angel Suarez (OU), *Colloborative inquiry-based learning.*

2018

01 Han van der Aa (VUA), *Comparing and Aligning Process Representations.*

02 Felix Mannhardt (TUE), *Multi-perspective Process Mining.*