

SOUND INTEGRATION OF PROCESS AND DECISION MODELS

KIMON BATOULIS

BUSINESS PROCESS TECHNOLOGY GROUP
HASSO PLATTNER INSTITUTE
DIGITAL ENGINEERING FACULTY
UNIVERSITY OF POTSDAM
POTSDAM, GERMANY

DISSERTATION
ZUR ERLANGUNG DES AKADEMISCHEN GRADES EINES
"DOCTOR RERUM NATURALIUM"
– DR. RER. NAT. –

DATE OF DEFENSE: 23/10/2019

November 2019

This work is licensed under a Creative Commons License:
Attribution 4.0 International.

This does not apply to quoted content from other authors.

To view a copy of this license visit

<https://creativecommons.org/licenses/by/4.0/>

Supervisor: Prof. Dr. Mathias Weske, University of Potsdam

Reviewers: Prof. Dr. Jan Vanthienen, KU Leuven, and

Prof. Dr. Marco Montali, Free University of Bozen-Bolzano

Kimon Batoulis: *Sound Integration of Process and Decision Models*,

© November 2019

Published online at the

Institutional Repository of the University of Potsdam:

<https://doi.org/10.25932/publishup-43738>

<https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-437386>

ABSTRACT

Business process management is an established technique for business organizations to manage and support their processes. Those processes are typically represented by graphical models designed with modeling languages, such as the Business Process Model and Notation (BPMN).

Since process models do not only serve the purpose of documentation but are also a basis for implementation and automation of the processes, they have to satisfy certain correctness requirements. In this regard, the notion of soundness of workflow nets was developed, that can be applied to BPMN process models in order to verify their correctness. Because the original soundness criteria are very restrictive regarding the behavior of the model, different variants of the soundness notion have been developed for situations in which certain violations are not even harmful.

All of those notions do only consider the control-flow structure of a process model, however. This poses a problem, taking into account the fact that with the recent release and the ongoing development of the Decision Model and Notation (DMN) standard, an increasing number of process models are complemented by respective decision models. DMN is a dedicated modeling language for decision logic and separates the concerns of process and decision logic into two different models, process and decision models respectively.

Hence, this thesis is concerned with the development of decision-aware soundness notions, i.e., notions of soundness that build upon the original soundness ideas for process models, but additionally take into account complementary decision models. Similar to the various notions of workflow net soundness, this thesis investigates different notions of decision soundness that can be applied depending on the desired degree of restrictiveness. Since decision tables are a standardized means of DMN to represent decision logic, this thesis also puts special focus on decision tables, discussing how they can be translated into an unambiguous format and how their possible output values can be efficiently determined.

Moreover, a prototypical implementation is described that supports checking a basic version of decision soundness. The decision soundness notions were also empirically evaluated on models from participants of an online course on process and decision modeling as well as from a process management project of a large insurance company. The evaluation demonstrates that violations of decision soundness indeed occur and can be detected with our approach.

ZUSAMMENFASSUNG

Das Prozessmanagement ist eine etablierte Methode für Unternehmen zur Verwaltung und Unterstützung ihrer Geschäftsprozesse. Solche Prozesse werden typischerweise durch graphische Modelle dargestellt, welche mit Modellierungssprachen wie etwa der Business Process Model and Notation (BPMN) erstellt werden.

Da Prozessmodelle nicht nur der Dokumentation der Prozesse dienen, sondern auch die Grundlage für deren Implementierung und Automatisierung sind, müssen sie bestimmte Korrektheitsanforderungen erfüllen. In dieser Hinsicht wurde der Begriff der Soundness eines Workflow-Netztes entwickelt, welcher auch auf BPMN-Prozessmodelle angewendet werden kann, um deren Korrektheit zu prüfen. Da die ursprünglichen Soundness-Kriterien sehr restriktiv bezüglich des Verhaltens des Modells sind, wurden zudem Varianten des Soundness-Begriffs entwickelt. Diese können in Situationen verwendet werden, in denen bestimmte Verletzungen der Kriterien tolerabel sind.

Diese Soundness-Begriffe berücksichtigen allerdings ausschließlich den Kontrollfluss der Prozessmodelle. Dies stellt ein Problem dar, weil viele Prozessmodelle heutzutage durch Entscheidungsmodelle ergänzt werden. In diesem Kontext ist die Decision Model and Notation (DMN) eine dedizierte Sprache zur Modellierung von Entscheidungen und unterstützt die Trennung von Kontrollfluss- und Entscheidungslogik.

Die vorliegende Dissertation befasst sich daher mit der Entwicklung von erweiterten Soundness-Begriffen, die sowohl Prozess- als auch Entscheidungsmodelle berücksichtigen. Ähnlich zu den bestehenden Soundness-Varianten, werden in dieser Arbeit Varianten des erweiterten Soundness-Begriffs untersucht, die je nach gewünschtem Restriktionsgrad angewendet werden können. Da Entscheidungstabellen eine in der DMN standardisierte Form sind, um Entscheidungslogik auszudrücken, fokussiert sich diese Arbeit insbesondere auf Entscheidungstabellen. So wird diskutiert wie DMN-Tabellen in ein eindeutiges Format übersetzt werden können und wie sich deren möglichen Rückgabewerte effizient bestimmen lassen.

Ferner beschreibt die Arbeit eine prototypische Implementierung, die das Prüfen einer elementaren Variante des erweiterten Soundness-Begriffs erlaubt. Die Begriffe wurden außerdem empirisch evaluiert. Dazu dienten zum einen Modelle von Teilnehmern eines Online-Kurses zur Prozess- und Entscheidungsmodellierung. Zum anderen wurden Modelle eines Versicherungsunternehmens analysiert. Die Evaluierung zeigt, dass Verstöße gegen den erweiterten Soundness-Begriff in der Tat auftreten und durch den hier beschriebenen Ansatz erkannt werden können.

PUBLICATIONS

Some ideas and figures have appeared previously in the following publications:

1. Kimon Batoulis, Andreas Meyer, Ekaterina Bazhenova, Gero Decker, and Mathias Weske. "Extracting Decision Logic from Process Models." In: *Advanced Information Systems Engineering*. Ed. by Jelena Zdravkovic, Marite Kirikova, and Paul Johannesson. Cham: Springer International Publishing, 2015, pp. 349–366. ISBN: 978-3-319-19069-3.
2. Kimon Batoulis, Stephan Haarmann, and Mathias Weske. "Various Notions of Soundness for Decision-Aware Business Processes." In: *Conceptual Modeling*. Ed. by Heinrich C. Mayr, Giancarlo Guizzardi, Hui Ma, and Oscar Pastor. Cham: Springer International Publishing, 2017, pp. 403–418. ISBN: 978-3-319-69904-2.
3. Kimon Batoulis, Alexey Nesterenko, Guenther Repitsch, and Mathias Weske. "Decision Management in the Insurance Industry: Standards and Tools." In: *Proceedings of the BPM 2017 Industry Track co-located with the 15th International Conference on Business Process Management (BPM 2017), Barcelona, Spain, September 10-15, 2017*. 2017, pp. 52–63.
4. Kimon Batoulis and Mathias Weske. "A Tool for Checking Soundness of Decision-Aware Business Processes." In: *Proceedings of the BPM Demo Track and BPM Dissertation Award co-located with 15th International Conference on Business Process Modeling (BPM 2017), Barcelona, Spain, September 13, 2017*. 2017.
5. Kimon Batoulis and Mathias Weske. "Soundness of Decision-Aware Business Processes." In: *Business Process Management Forum*. Ed. by Josep Carmona, Gregor Engels, and Akhil Kumar. Cham: Springer International Publishing, 2017, pp. 106–124. ISBN: 978-3-319-65015-9.
6. Kimon Batoulis and Mathias Weske. "A Tool for the Uniqueification of DMN Decision Tables." In: *Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2018 co-located with 16th International Conference on Business Process Management (BPM 2018), Sydney, Australia, September 9-14, 2018*. 2018, pp. 116–119.

7. Kimon Batoulis and Mathias Weske. "Disambiguation of DMN Decision Tables." In: *Business Information Systems*. Ed. by Witold Abramowicz and Adrian Paschke. Cham: Springer International Publishing, 2018, pp. 236–249. ISBN: 978-3-319-93931-5. (Best Paper Award).
8. Stephan Haarmann, Kimon Batoulis, and Mathias Weske. "Compliance Checking for Decision-Aware Process Models." In: *Business Process Management Workshops - BPM 2018 International Workshops, Sydney, NSW, Australia, September 9-14, 2018, Revised Papers*. 2018, pp. 494–506.

In addition to above publications, the author of this thesis was also involved in the following research indirectly contributing to the thesis:

9. Kimon Batoulis, Rami-Habib Eid-Sabbagh, Henrik Leopold, Mathias Weske, and Jan Mendling. "Automatic Business Process Model Translation with BPMT." In: *Advanced Information Systems Engineering Workshops*. Ed. by Xavier Franch and Pnina Soffer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 217–228. ISBN: 978-3-642-38490-5.
10. Kimon Batoulis. "Proactive Decision Support During Business Process Execution." In: *Joint Proceedings of the 1st International Workshop on Modeling Inter-Organizational Processes and 1st International Workshop on Event Modeling and Processing in Business Process Management co-located with Modellierung 2014, Vienna, Austria, March 19, 2014*. 2014, pp. 35–41.
11. Andreas Meyer, Luise Pufahl, Kimon Batoulis, Sebastian Kruse, Thorben Lindhauer, Thomas Stoff, Dirk Fahland, and Mathias Weske. "Automating Data Exchange in Process Choreographies." In: *Advanced Information Systems Engineering*. Ed. by Matthias Jarke, John Mylopoulos, Christoph Quix, Colette Rolland, Yannis Manolopoulos, Haralambos Mouratidis, and Jennifer Horkoff. Cham: Springer International Publishing, 2014, pp. 316–331. ISBN: 978-3-319-07881-6.
12. Andreas Meyer, Luise Pufahl, Kimon Batoulis, Dirk Fahland, and Mathias Weske. "Automating data exchange in process choreographies." In: *Information Systems* 53 (2015), pp. 296–329. ISSN: 0306-4379.
13. Han van der Aa, Henrik Leopold, Kimon Batoulis, Mathias Weske, and Hajo A. Reijers. "Integrated Process and Decision Modeling for Data-Driven Processes." In: *Business Process Management Workshops*. Ed. by Manfred Reichert and Hajo A. Reijers. Cham: Springer International Publishing, 2016, pp. 405–417. ISBN: 978-3-319-42887-1.

14. Kimon Batoulis, Anne Baumgraß, Nico Herzberg, and Mathias Weske. "Enabling Dynamic Decision Making in Business Processes with DMN." In: *Business Process Management Workshops*. Ed. by Manfred Reichert and Hajo A. Reijers. Cham: Springer International Publishing, 2016, pp. 418–431. ISBN: 978-3-319-42887-1.
15. Adriatik Nikaj, Kimon Batoulis, and Mathias Weske. "REST-Enabled Decision Making in Business Process Choreographies." In: *Service-Oriented Computing*. Ed. by Quan Z. Sheng, Eleni Stroulia, Samir Tata, and Sami Bhiri. Cham: Springer International Publishing, 2016, pp. 547–554. ISBN: 978-3-319-46295-0.
16. Luise Pufahl, Sankalita Mandal, Kimon Batoulis, and Mathias Weske. "Re-evaluation of Decisions Based on Events." In: *Enterprise, Business-Process and Information Systems Modeling*. Ed. by Iris Reinhartz-Berger, Jens Gulden, Selmin Nurcan, Wided Guédria, and Palash Bera. Cham: Springer International Publishing, 2017, pp. 68–84. ISBN: 978-3-319-59466-8.
17. Stephan Haarmann, Kimon Batoulis, Adriatik Nikaj, and Mathias Weske. "DMN Decision Execution on the Ethereum Blockchain." In: *Advanced Information Systems Engineering*. Ed. by John Krogstie and Hajo A. Reijers. Cham: Springer International Publishing, 2018, pp. 327–341. ISBN: 978-3-319-91563-0.
18. Alaaeddine Yousfi, Kimon Batoulis, and Mathias Weske. "Achieving Business Process Improvement via Ubiquitous Decision-Aware Business Processes." In: *ACM Trans. Internet Technol.* 19.1 (Jan. 2019), 14:1–14:19. ISSN: 1533-5399.

ACKNOWLEDGMENTS

This dissertation is as much a work of mine as it is of my supervisor Mathias Weske, who through his guidance and critical feedback stimulated thoughts and ideas that would have otherwise stayed dormant. I am grateful for the huge variety of experiences I could gain during my time as a research assistant in his group.

Many thanks to Marco Montali and Jan Vanthienen for reviewing this thesis. Their work and activity in the BPM community have been an inspiration for me and served as prime examples for excellent research.

Writing a PhD thesis is a decision that is not made easily. I was lucky to be introduced to scientific work by Rami Eid-Sabbagh and Henrik Leopold who co-authored my first paper with me and showed me the beauty of BPM research.

Although this thesis has a single author, there are many fine researchers whose input and feedback were vital for the completion of this work. I especially want to express my gratitude to my colleagues Stephan Haarmann and Adriatik Nikaj. Stephan for his constant interest in and contribution to my research work during his time as a master's student, and for his consistent availability for discussions as a colleague at BPT. Tiku for his undying interest and support in the most intricate problems of my research. His patience and altruism are fascinating.

I'd like to thank all of my (ex-)colleagues for creating a great atmosphere at the BPT group. Kiarash Diba, Stephan Haarmann, Sven Ihde, Jan Ladleif, Luise Pufahl and Simon Remy for proofreading this thesis. Sankalita Mandal and Tiku for sharing the suffering of completing a PhD thesis.

Finally, I want to express my appreciation for my family and friends. My parents — they have helped me make lots of decisions. Some of them wrong, many of them right.

CONTENTS

I BACKGROUND

1	INTRODUCTION	3
1.1	Research Objective	4
1.2	Contributions	4
1.3	Structure of the Thesis	6
2	FOUNDATIONS AND RELATED WORK	9
2.1	Business Process Management	9
2.1.1	Business Process Lifecycle	9
2.1.2	Business Process Models	11
2.1.3	Verification of Business Process Models	23
2.2	Business Decision Management	29
2.2.1	DMN Decision Models	30
2.2.2	DMN Decision Table Hit Policies	32
2.2.3	Formalization of DMN Decision Models	33
2.2.4	Decision Table Analysis	34
2.3	Integration of Process and Decision Management	35
2.4	Related Work	38
2.4.1	Integration of Processes and Rules	38
2.4.2	Integration of Processes and Decisions	40
2.4.3	DMN Decision Model Analysis	43
2.4.4	Process Verification with Data	44

II FORMAL FRAMEWORK FOR PROCESSES AND DECISIONS

3	ON THE SEPARATION OF CONCERNS OF PROCESSES AND DECISIONS	49
3.1	Process and Decision Modeling in the Real World	49
3.2	Control-Flow-Based Decision Patterns	51
3.2.1	P ₁ —Single Split Gateway	52
3.2.2	P ₂ —Sequence of Split Gateways (Decision Tree)	53
3.2.3	P ₃ —Sequence of Split Gateways Separated by an Activity	54
3.3	Statistics about the Decision Patterns	55
4	INPUT-OUTPUT BEHAVIOR OF DMN DECISION TABLES	59
4.1	Problem Statement	59
4.2	Decision Table Uniqueification	60
4.3	Decision Tables as Functions	66
4.4	Maximum Number of Outputs of a Decision Table	68
5	FORMALIZING DECISION-AWARE PROCESS MODELS	75
5.1	Simple Colored Petri Net Mapping	75
5.1.1	Merging Decision Tables	75
5.1.2	Mapping Decision Fragments to CPNs	77
5.2	Abstract Colored Petri Net Mapping	78
5.2.1	Symbolic Abstraction	78

5.2.2	Mapping Decision Fragments to Abstract CPNs	81
5.2.3	Assumptions	84
5.3	Colored Workflow Nets and Their Soundness	85
5.4	Concurrent Execution of Decision Fragments	91
5.5	Summary and Discussion	92
III SOUNDNESS OF DECISION-AWARE BUSINESS PROCESSES		
6	STATELESS DECISION SOUNDNESS	97
6.1	Motivation and Problem Statement	97
6.2	Structural Consistency	98
6.3	Behavioral Consistency	99
6.3.1	Decision Deadlock Freedom	99
6.3.2	Dead Branch Absence	101
6.4	Stateless Decision-aware Soundness	102
6.5	Discussion and Shortcomings	103
7	STATE-BASED DECISION SOUNDNESS	105
7.1	Using State Information for Soundness Checking	105
7.2	State-based Decision Deadlock Freedom	109
7.3	State-based Dead Branch Absence	111
8	A TAXONOMY FOR DECISION SOUNDNESS	113
8.1	Relationship between the Stateless and State-based Decision Soundness Criteria	113
8.1.1	DDF \implies SB-DDF	113
8.1.2	SB-DBA \implies DBA	114
8.2	Various Notions of Decision Soundness	115
8.2.1	Decision Soundness	115
8.2.2	Relaxed Decision Soundness	116
8.2.3	Weak Decision Soundness	117
8.2.4	Easy Decision Soundness	118
8.2.5	Lazy Decision Soundness	119
8.3	Summary and Discussion	120
IV EVALUATION AND CONCLUSION		
9	EVALUATION	125
9.1	Prototypical Implementation	125
9.1.1	Extended Camunda Tool	125
9.1.2	Runtime Analysis	126
9.2	Empirical Evaluations	130
9.2.1	Academic Models	131
9.2.2	Industry Use Case	134
9.3	Compliance Checking of Decision-aware Processes	136
10	CONCLUSION	139
10.1	Summary	139
10.2	Limitations and Future Work	140
BIBLIOGRAPHY		143

LIST OF FIGURES

Figure 1.1	Overview of the contributions of this thesis	5
Figure 2.1	Business process life cycle (cf. [53, p. 12])	10
Figure 2.2	Petri net consisting of four places and four transitions	12
Figure 2.3	Reachability graph for the Petri net shown in Figure 2.2	13
Figure 2.4	A simple colored Petri net checking the gender of a person	16
Figure 2.5	BPMN notational elements for business process diagrams considered in this thesis	19
Figure 2.6	Train ticket booking process derived from booking tickets with Deutsche Bahn, modeled as a BPMN process diagram	22
Figure 2.7	UML class diagram for the <i>Booking</i> data object in Figure 2.6	22
Figure 2.8	Mapping of BPMN task, events and gateways to Petri nets (cf. [30])	24
Figure 2.9	A workflow net that is not sound, but relaxed sound	26
Figure 2.10	A weak sound workflow net	26
Figure 2.11	A workflow net that is neither sound, nor weak sound, nor relaxed sound, but lazy sound	27
Figure 2.12	A workflow net that is only easy sound	28
Figure 2.13	Various notions of soundness and their relationships [44]	28
Figure 2.14	Reachability graph for the Petri net shown in Figure 2.9	29
Figure 2.15	An abstract DMN decision requirements diagram	31
Figure 2.16	An example of a DMN decision table	31
Figure 2.17	DMN decision table to determine a credit rating	35
Figure 2.18	Decision table as a set of hyperrectangles corresponding to the rules in the table in Figure 2.17	35
Figure 2.19	An abstract decision fragment conforming to Definition 20	36
Figure 2.20	Decision model referenced by the task <i>Manage discount</i> in Figure 2.6	37
Figure 2.21	Decision model referenced by the task <i>Manage special offer</i> in Figure 2.6	37

Figure 2.22	Decision task directly followed by another task reading the output of the decision	37	
Figure 2.23	Classification of related work	38	
Figure 3.1	Misuse of BPMN for decision logic modeling		50
Figure 3.2	Process fragment representing a split gateway with more than 2 outgoing edges	52	
Figure 3.3	Process fragment representing a sequence of split gateways that represents a decision tree		53
Figure 3.4	Process fragment representing a sequence of split gateways separated by an activity	55	
Figure 3.5	Frequencies of patterns P_1 – P_3 in real world process models	56	
Figure 4.1	DMN decision table to determine a credit rating	59	
Figure 4.2	Decision table as a set of hyperrectangles corresponding to the rules of the table in Figure 4.1		61
Figure 4.3	Line being swept through the second dimension in the interval $[0, 10)$	63	
Figure 4.4	Line being swept through the second dimension in the interval $[10, 20)$	63	
Figure 4.5	Set of <i>matchingRules</i> found so far before and after merge	64	
Figure 4.6	Decision table as a set of hyperrectangles	66	
Figure 4.7	Uniqueified table derived from the table in Figure 4.1	66	
Figure 4.8	Decision table with three rules and two different output values	67	
Figure 4.9	Example of a decision table with one input column, and its geometric interpretation	69	
Figure 4.10	Another example of a decision table with one input column, and its geometric interpretation		69
Figure 4.11	Induction base case	70	
Figure 4.12	Induction hypothesis	70	
Figure 4.13	Induction step—Option 1	71	
Figure 4.14	Induction step—Option 2	71	
Figure 4.15	Induction step—Option 3	72	
Figure 4.16	Induction step—Option 4	72	
Figure 5.1	Decision requirements diagram with more than one decision	76	
Figure 5.2	Decision tables associated with the decisions in Figure 5.1	76	
Figure 5.3	Equivalent decision model with merged decision tables	76	
Figure 5.4	Example of a mapping of a decision task that references the decision table in 5.4a to a colored Petri net	77	

Figure 5.5	Colored Petri net corresponding to the table in Figure 5.4a, employing symbolic abstraction	80
Figure 5.6	Reachability graph of the colored Petri net in Figure 5.5	81
Figure 5.7	A generic decision fragment with its associated decision table	82
Figure 5.8	Colored Petri net corresponding to the decision fragment in Figure 5.7, employing symbolic abstraction	83
Figure 5.9	Decision-aware process model that references the table in Figure 5.4a and that can be translated to a colored workflow net	86
Figure 5.10	Colored Petri net mapping of the decision-aware process model in Figure 5.9	89
Figure 5.11	Reachability graph of colored workflow net in Figure 5.10	90
Figure 5.12	Decision tasks being executed concurrently	91
Figure 5.13	Partial mapping of the process model in Figure 5.12	92
Figure 6.1	Train ticket booking process derived from booking tickets with Deutsche Bahn, modeled as a BPMN process diagram	97
Figure 6.2	Decision tables referenced by the decision tasks of the process model in Figure 6.1	97
Figure 6.3	Different possibilities of covering table outputs by edge conditions	101
Figure 6.4	A decision fragment that violates the <i>dead branch absence</i> criterion	102
Figure 6.5	Decision task directly followed by another task reading the output of the decision	103
Figure 7.1	Train ticket booking process derived from booking tickets with Deutsche Bahn, repeated for convenience	105
Figure 7.2	Decision tables referenced by the decision tasks of the process model in Figure 7.1, repeated for convenience	105
Figure 7.3	Simplified version of the process model in Figure 7.1	106
Figure 7.4	Colored Petri net mapping of the decision-aware process model in Figure 7.3	107
Figure 7.5	Reachability graph of the colored Petri net in Figure 7.4	108
Figure 7.6	Sketch of a CPN decision fragment to visualize Definitions 34 and 35	110
Figure 7.7	Adapted version of the process model in Figure 7.3	111

Figure 8.1	Various notions of (decision) soundness	115
Figure 8.2	Non-decision sound variant of the process model in Figure 7.3	116
Figure 8.3	Decision table called by <i>Manage special offer</i>	116
Figure 8.4	Relaxed decision sound variant of the process model in Figure 7.3	116
Figure 8.5	Easy decision sound variant of the process model in Figure 7.3	119
Figure 8.6	Decision tables referenced by the decision tasks of the process model in Figure 8.5	119
Figure 8.7	Lazy decision sound variant of the process model in Figure 7.3	120
Figure 8.8	Various notions of decision soundness annotated with their respective conditions as defined in Section 8.2	120
Figure 9.1	View of the tool after checking soundness of the displayed decision-aware process model	127
Figure 9.2	View of the tool after uniqueifying the table shown in Figure 9.1	128
Figure 9.3	Averaged execution times in seconds for a set of 1000 multi-hit tables with up to 50 rows and 30 columns	129
Figure 9.4	Total number of overlaps found in the tables	130
Figure 9.5	Modeling exercise given to openHPI course participants	131
Figure 9.6	Model for the billing process described in Figure 9.5	132
Figure 9.7	Decision table for the billing decision described in Figure 9.5	132
Figure 9.8	Submission by a course participant for the modeling task described in Figure 9.5	133
Figure 9.9	Decision table associated with the decision fragment in Figure 9.8	133
Figure 9.10	Example of decision-aware process model found in the analyzed Pega project	135
Figure 9.11	Decision table associated with the gateway in Figure 9.10	135
Figure 9.12	Example for a Pega decision table taken from the analyzed project	135
Figure 9.13	Number of decision tables of the Pega project that are complete according to DMN (stateless) and Pega (state-based)	136
Figure 9.14	The train ticket booking process model, repeated for convenience	137

LIST OF TABLES

Table 2.1	Comparison of the (in)consistency criteria described in [103] and in this thesis (based on [77, 80]) 43
Table 3.1	Statistics on the usage of split gateways in the analyzed model collections 56
Table 9.1	Number and percentage of sound and unsound decision-aware process models designed by the MOOC course participants 133

ACRONYMS

BPM	Business process management
BPMN	Business Process Model and Notation
BR	Business rule
CMMN	Case Management Model and Notation
CPN	Colored Petri net
DBA	Dead branch absence
DDF	Decision deadlock freedom
DMN	Decision Model and Notation
DPN	Data Petri net
DRD	Decision requirements diagram
FEEL	Friendly Enough Expression Language
MOOC	Massive open online course
OMG	Object Management Group
PAIS	Process-aware information system
SB-DBA	State-based dead branch absence
SB-DDF	State-based decision deadlock freedom
SBVR	Semantics of Business Vocabulary and Rules

S-FEEL	Simplified Friendly Enough Expression Language
SRML	Simple Rule Markup Language
UML	Unified Modeling Language
WFD net	Workflow net with data

Part I

BACKGROUND

INTRODUCTION

Business process management (**BPM**) has become a major topic, for academia and industry alike [53]. BPM's core artifacts are business processes, which it aims to design, analyze, execute and improve. Processes are everywhere and every organization has processes [100]. The fundamental components of a business process are its activities that are executed to achieve a certain business objective [13]. At the same time, most processes do not simply consist of a single stream of activities. Activities may be executed concurrently and there may be points in the process at which one of several alternative branches must be chosen in order to continue. This choice is the outcome of a decision, which is "the *act* of choosing among multiple possible options" [114, p. 28].

The industry standard for designing business process models is the Business Process Model and Notation (**BPMN**) [59]. This modeling language is, however, not suited to represent decision logic as it focusses on the activities that need to be performed and not the decisions to be made. Traditionally, decision logic could be maintained in business rules—simple if-then statements [52]. Unfortunately, their interplay with process models was mostly left unclear. In this regard, the Decision Model and Notation (**DMN**) [114] offers an alternative in that it is a decision modeling language explicitly designed to be used complementary to BPMN to express a process' decision logic. This conjunctive use of process and decision models led to the notion of decision-aware business processes [35].

The prime focus of a process model, however, has always been the control-flow perspective, its documentation, implementation and automation. In particular with respect to implementation and automation, special attention has to be given to the correctness of the process model. Therefore, the notion of soundness of workflow nets was developed, that can be applied to process models in order to verify their correctness [16]. The fact that this notion only considers the control-flow perspective poses a problem considering the rising interest in and use of decision-aware process models. This is because decision models have an influence on the control-flow of the process and as a result also on its soundness or correctness. For example, a decision model can cause a process to deadlock, making it impossible to terminate the process properly. Also, it may lead to dead branches in the process, so that the activities that are intended for those branches can never be executed.

Hence, the existing soundness notion has to be lifted to the setting of decision-aware process models to allow for a sound integration of process and decision models. Existing approaches such as [103] tackling a consistent integration of the two kinds of models fail to cover the entire spectrum of soundness and do not provide a formal verification method, whereas the work described in [107] is limited to a certain type of DMN decision tables.

1.1 RESEARCH OBJECTIVE

The objective of this thesis is to provide a formal basis and a method for the verification of the soundness of a process model that is associated with one or more decision models. This requires the definition of a corresponding notion of soundness of a decision-aware process model, also called decision soundness, and a formalization of decision-aware process models such that decision soundness can be formally verified.

This thesis also acknowledges the fact that decision models tend to be reused by multiple process models of an organization, making the same or similar decisions during execution. Therefore, a decision model will not perfectly fit with every process model it is associated with. In other words, it may contain logic that is relevant for some processes but not for others. Accordingly, besides the basic notion of decision soundness, it is necessary to formulate various relaxed notions of decision soundness, such that a decision-aware process model can be denoted decision sound for various situations in which it does not perfectly fit with the decision model.

1.2 CONTRIBUTIONS

In reference to the research objective delineated above, this section explains the contributions of this thesis. Overall, the contribution is a set of soundness notions of decision-aware business process models in conjunction with a method to verify those notions for a given process model. This main contribution entails several more fine-grained contributions which are described in the following along Figure 1.1.

- *The need for a separation of processes and decisions:* Based on the analysis of nearly 1000 real world process models, it is revealed that decision logic is often hard-coded in process models via three frequently occurring control-flow-based decision patterns. We argue that this violates the separation of concerns paradigm, substantiating the need to express a process' decision logic in a dedicated decision model. This is illustrated at the top of Figure 1.1, where a process with decisions should be modeled as a process *and* decision model, rather than a process model alone.

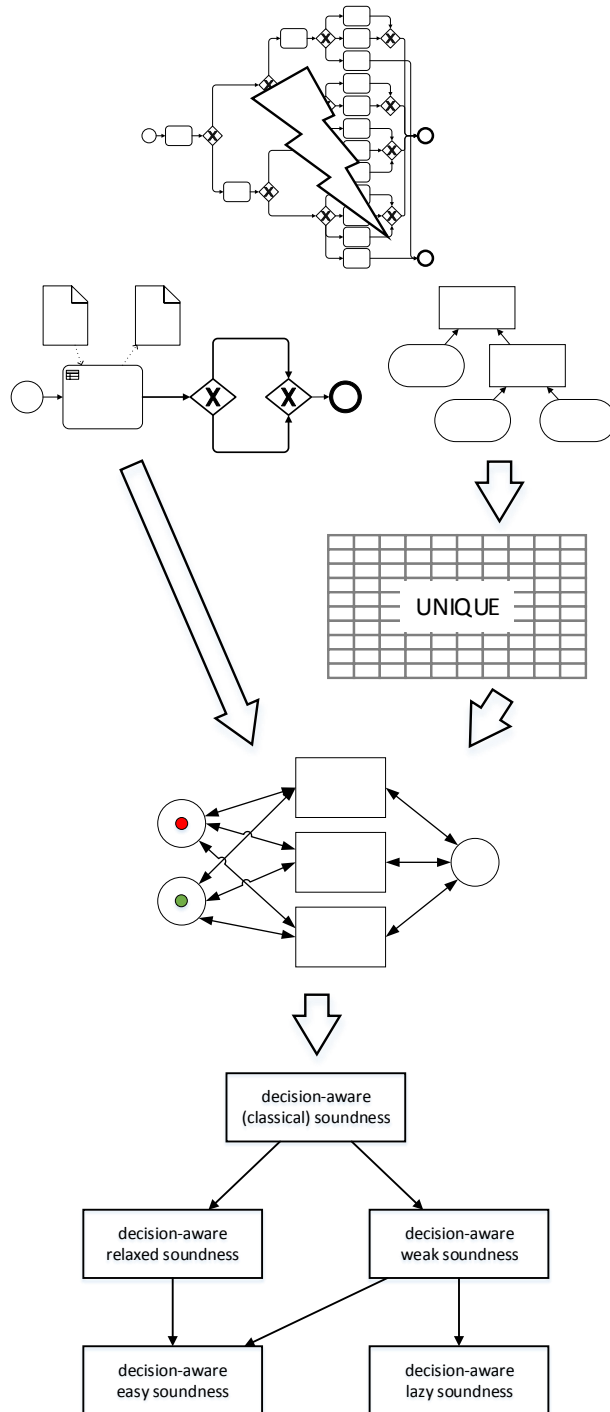


Figure 1.1: Overview of the contributions of this thesis

- *Uniqueification of DMN decision tables:* In accordance with the recognition that decision tables are the standard means of expressing decision logic and the fact that a process communicates with decisions via their inputs and outputs, we conduct an analysis of the input/output behavior of DMN decision tables. As a result, we describe an algorithm that transforms any DMN table into a standardized (or “uniqueified”) representation that not only increases understandability of the table but also sets the foundations for analyzing that table in association with a process model in regard to soundness. Moreover, we discuss the maximum number of outputs of a DMN table. This is illustrated in Figure 1.1 by the small arrow leading from the decision model at the top to the table labeled *unique*.
- *Formalization of decision-aware process models:* In order to verify the soundness of a process model associated with a decision model, an appropriate formalization is presented. This also includes an abstraction technique to deal with the fact that including decisions in the verification leads to infinitely many possible states of the process. This is illustrated in Figure 1.1 by the two arrows leading from the process model and the decision table to the colored Petri net.
- *Basic decision soundness criteria:* We define basic structural and behavioral criteria that can actually be checked directly on the process model, without the need to analyze the formal representation of the model as a colored Petri net.
- *Various notions of decision soundness:* Different variations and combinations of the decision soundness criteria are defined. This leads to different notions of decision soundness. Their relationship is shown and we demonstrate how to verify them based on the colored Petri net representation of the decision-aware process model. This is illustrated in Figure 1.1 by the arrow leading from the colored Petri net to the graph that shows the different decision soundness notions and their relationships.

1.3 STRUCTURE OF THE THESIS

This thesis is divided into four parts, whose contents will be described in the following:

PART I. After having described the research objective and contributions, the first part of this thesis continues with laying the foundations of the presented work in Chapter 2. This includes the description of business process management and business decision management and their relationship. Additionally, process and decision modeling languages are introduced alongside the running

example of this thesis. Moreover, process and decision analysis techniques are described, and the part concludes with a detailed discussion of related work.

PART II. The first main part of this work builds a formal framework for processes and decisions and their analysis. It starts off by presenting the empirical analysis of a large collection of real world process models in Chapter 3. The analysis shows that process models tend to violate the separation of concerns by encoding decision logic using three frequently occurring control-flow-based decision patterns.

Chapter 4 deals with the input/output behavior of DMN decision tables. It presents an algorithm that transforms any DMN table into a standardized/uniqueified table, which is a prerequisite for decision soundness analysis in general and for the formalization of a decision-aware process model in particular. Moreover, it discusses how decision tables can be interpreted as functions and proves what the maximum number of outputs of a table with one input column is.

Finally, Chapter 5 shows how a decision-aware process model can be formalized as a colored Petri net, employing an abstraction technique to deal with the infinity introduced by including decisions. Additionally, colored workflow nets and their soundness are defined.

PART III. The second main part of this thesis introduces the soundness notions for decision-aware process models. In Chapter 6 basic criteria for structural and behavioral consistency between a process and a decision model are defined. The basic behavioral criteria have certain limitations, especially in the light of the reusability of decision models.

Therefore, in Chapter 7 those criteria are refined to take the context of the process model into account with which a decision model is associated. Moreover it is shown how the refined criteria can be verified formally.

Lastly, in Chapter 8 different variations and combinations of the decision soundness criteria are defined and organized into a taxonomy that also specifies their relationships.

PART IV. The last part of this work is mainly concerned with the evaluation of the concepts and ideas presented in this thesis in Chapter 9. First, a prototypical implementation for checking the basic behavioral decision soundness criteria is described and its performance is evaluated. Afterwards, the decision soundness criteria are empirically evaluated based on two real world sets of decision-aware process models. Finally, we show that the formalization of decision-aware process models can also be applied to

another kind of verification, namely compliance checking.
Chapter 10 concludes the thesis with a summary and a discussion of limitations and potential directions for future work.

This chapter provides the background of this thesis and establishes the formal concepts required for this work. The topic of Section 2.1 is business process management, including the description of various process modeling languages and the verification of the corresponding process models. Afterwards, business decision management is addressed in Section 2.2, introducing decision models, their formalization and the analysis of decision tables. Section 2.3 discusses the integration of process and decision management, while Section 2.4 concludes this chapter with a discourse on related work.

2.1 BUSINESS PROCESS MANAGEMENT

Business process management (BPM) is an instrument to achieve a process oriented view of an organization's operations. Since organizations run a business, BPM aims to arrange these operations into business processes. Business processes are defined in many different ways [20, 46, 53, 90, 100]. Their common denominator is the concept of a set of activities performed to achieve a business goal. Therefore, the definition given in [53, p. 5] by Weske provides a good summary by stating that a business process "consists of a set of activities that are performed in coordination in an organizational and technical environment" to "jointly realize a business goal".

2.1.1 *Business Process Lifecycle*

Since business processes are executed in and interact with the real world which is subject to continuous change, the business processes themselves also need to be adapted continuously. This constant adaptation can be described by a cyclic process called business process lifecycle which is illustrated in Figure 2.1. According to this life cycle each business process repeatedly traverses four phases, each of which consists of different activities conducted regarding the process.

The first phase is the *design and analysis* phase. The key activity in this phase is the scoping of business processes, since usually an organization is already conducting many activities but the correlation of these activities to well defined business processes is not clear. The identified business processes can then be represented in corresponding process models. This enables the documentation, verification and validation of the processes and their models. Since process models are primarily graphical models, they are the basis to exchange infor-

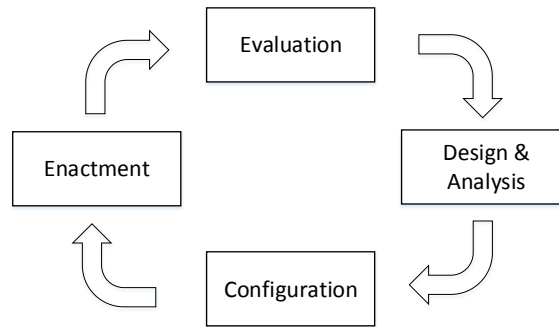


Figure 2.1: Business process life cycle (cf. [53, p. 12])

mation and knowledge about them. Furthermore, process models can be verified for certain correctness properties using formal techniques. For example, it can be asserted that the model and therefore the underlying process is free of deadlocks. Finally, the model should be designed in such a way that the process achieves the intended objective. Hence, validation techniques such as simulation can be applied to find out how the process would behave in the real world.

Having completed the *design and analysis* phase the business process enters the *configuration* phase. In this phase the business process is implemented in the organization. The implementation may or may not be realized in an IT system. In a simple case the process may just be implemented by assigning personnel to carry out the necessary steps in the intended order. More advanced implementations may have IT systems in place that support the execution of the business process. Even a so called process-aware information system (PAIS) may be used. These are systems that are more oriented towards processes than towards data, the latter having been the common case traditionally [53]. A typical PAIS supports the execution of the process based on its model. For this purpose, the graphical model must be enhanced with additional technical information such as a mechanism to allocate resources to activities and the handling of data operations and service calls.

In the *enactment* phase the business process is finally executed in its environment. In the course of this, execution data will be produced. This data may be used to provide monitoring information on the current state of the process during its enactment. For example, the process model can be enhanced to visualize which activity is currently being executed or how much time each activity took so far. Also, the collected data can be utilized in the next phase of the life cycle.

The last phase is concerned with the *evaluation* of the business process. Since there is always a gap between the process designed as a model and the process executed in the real world, analysts can use the collected execution data to examine if the model and the data are properly aligned or if there were any deviations. Moreover, bot-

tlenecks can be identified and eliminated, for example by assigning more resources to certain tasks or by introducing concurrency.

2.1.2 Business Process Models

According to Stachowiak [3] any model is characterized by three features. The mapping feature is given by the fact that every model is accompanied by an original—the entity being modeled—that may already exist, be in planning, or just be fictive. The abstraction feature expresses that the model does not represent every detail of the original but simplifies and abstracts in some aspects. Finally, the pragmatic feature says that the model can replace the original in certain situations, namely those that the model is made for.

Business process models are no different. They represent business processes that may or may not be in operation already. They also abstract from the original by not specifying everything in full detail. For example, they may omit the exact type of document that is updated during the process (e.g., an Excel sheet). Finally, they can be used for different purposes such as the documentation of an existing business process or as a blueprint for the implementation of a planned process.

Over the years, many languages have been designed or used for the purpose of modeling business processes. A good overview of them is given in [40]. In the following two sections more detailed descriptions of Petri nets and BPMN are given.

2.1.2.1 Petri Nets

Petri nets, invented by Carl Adam Petri and fully specified in [1], are simple yet powerful graphs originally designed to describe chemical processes. However, they are also well suited to model business processes because of their ability to represent concurrency. Moreover, they have a precise mathematical formalization making them available for correctness verification [15] and other analysis techniques [42, 45].

A Petri net is a bipartite graph composed of places, transitions, and directed edges between the two. Places are represented as circles and transitions as rectangles. From this, the simple definition of a Petri net follows (based on [72]):

Definition 1 (Petri net). A Petri net is a triplet $N = (P, T, F)$ where

- P is a finite set of places,
- T is a finite set of transitions, with $T \cap P = \emptyset$, and
- $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation.

◇

For any transition $t \in T$, the input places of t are denoted by $\bullet t = \{p \mid (p, t) \in F\}$, whereas its output places are given by $t\bullet = \{p \mid (t, p) \in F\}$. Places can passively hold tokens, while transitions actively consume tokens from their input places and produce tokens on their output places. Such tokens are represented by black circles.

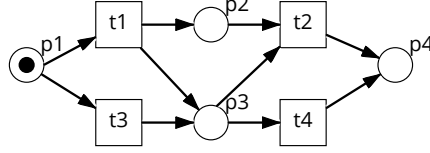


Figure 2.2: Petri net consisting of four places and four transitions

Figure 2.2 shows a Petri net with four places and four transitions. Place $p1$ holds a token and is the input place of transition $t1$, whereas places $p2$ and $p3$ are the output places of $t1$.

Petri nets can represent the states and state transitions of a system. The state is given by the distribution of tokens across the places, called the marking of the Petri net. State transitions are achieved by the *firing* of transitions. A transition can fire as soon as it is enabled, which requires that there is a token in every input place of that transition. Therefore, the input places of a transition can be interpreted as conditions that are fulfilled as soon as the places contain a token. Upon firing, the transition consumes (i.e., removes) one token from each of its input places and produces one token on each output place. Given that the transition's set of input places is not equal to its set of output places, the marking of the Petri net then changes. With that, a state transition of the underlying system is represented.

Formally, the marking of a Petri net is given by the function $M : P \rightarrow \mathbb{N}_0$, assigning a natural number including 0 to each place of the Petri net. For example, the marking of the Petri net in Figure 2.2 is given by

$$\forall p \in P : M(p) = \begin{cases} 1 & \text{if } p = p1 \\ 0 & \text{otherwise} \end{cases}$$

This can be abbreviated by simply listing all places that hold at least one token and indicating the number of tokens of each place as a corresponding superscript. Therefore, in the example the marking can be expressed as $[p1^1]$.

With the help of the marking function M , it is possible to clearly define the enablement and firing of transitions (based on [72]):

Definition 2 (Petri net transition firing rule). Let $N = (P, T, F)$ be a Petri net and M a marking. A transition $t \in T$ is enabled if and only if $\forall p \in \bullet t : M(p) \geq 1$. The firing of t changes the marking M to M' , denoted as $M \xrightarrow{t} M'$, where $\forall p \in \bullet t : M'(p) = M(p) - 1 \wedge \forall p \in t\bullet :$

$M'(p) = M(p) + 1$. Furthermore, $M_1 \xrightarrow{*} M_n$ indicates that there is a firing sequence of transitions t_1, t_2, \dots, t_{n-1} such that $M_i \xrightarrow{t_i} M_{i+1}$, for $1 \leq i < n$. The marking M' is called reachable from M if and only if $M \xrightarrow{*} M'$. \diamond

Given a Petri net and some initial marking for that net, one can construct a graph consisting of all of its reachable markings, called the *reachability graph*. This graph visualizes the possible reachable markings starting from the initial marking, and which transition of the Petri net is responsible for changing from one marking to another. Formally, it is defined as follows (based on [53]):

Definition 3 (Petri net reachability graph). Let $N = (P, T, F)$ be a Petri net, M a marking, and M_I the initial marking. A directed graph $G = (V, E, l)$ is the reachability graph of N if:

- $V = \{M \mid M_I \xrightarrow{*} M\}$ is the set of vertices corresponding to the reachable markings of the Petri net,
- $E = \{(M, t, M') \in V \times T \times V \mid M \xrightarrow{t} M'\}$, is the set of edges such that there is an edge for each transition leading from one marking to another, and
- $l : E \rightarrow T, e \mapsto e^2$, where $e^i, i \in \mathbb{N}_{>0}$ refers to the i th element of the tuple e , is the labeling function assigning a label to each edge corresponding to the transition of that edge.

\diamond

Figure 2.3 shows the reachability graph corresponding to the Petri net in Figure 2.2. For example, it shows that starting from the initial marking holding a token in $p1$ it is possible to reach the marking $[p2, p3]$ by firing transition $t1$, and that after firing $t4$ from that marking, no other marking can be reached.

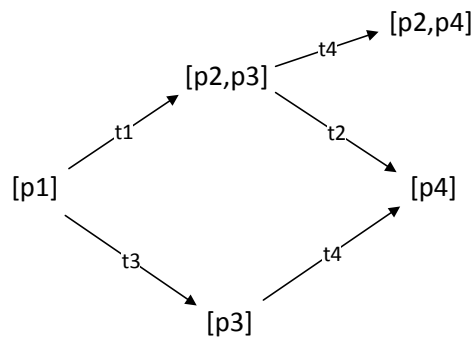


Figure 2.3: Reachability graph for the Petri net shown in Figure 2.2

Petri nets can be used as a modeling language for business processes. The activities of the process are represented as transitions in the Petri net and the current state of the process is reflected by

the net's marking. However, business processes usually have a distinguished start and end state. To impose this property on the corresponding Petri net, the notion of a workflow net was proposed [15]:

Definition 4 (Workflow net). Let $N = (P, T, F)$ be a Petri net. N is a workflow net if and only if:

- $(\exists i \in P, \neg \exists t \in T : i \in t \bullet) \wedge \forall p \in P : (\neg \exists t \in T : p \in t \bullet) \implies (p = i),$
- $(\exists o \in P, \neg \exists t \in T : o \in \bullet t) \wedge \forall p \in P : (\neg \exists t \in T : p \in \bullet t) \implies (p = o),$ and
- the Petri net $N' = (P, T \cup \{t'\}, F \cup \{(o, t'), (t', i)\})$ is strongly connected, i. e., every pair of nodes (places and transitions) of N' is connected via a directed path.

◇

The first two conditions require that the Petri net contains a single input place and a single output place respectively. This reflects the fact that the execution of a business process has a distinguished start condition and a distinguished end result. The third condition makes sure that the modeled process does not contain conditions or tasks that are not connected to the rest of the process.

2.1.2.2 Colored Petri Nets

While Petri nets serve as a simple and precise way to express business processes, they have a major drawback. The tokens that are consumed and produced by the net's transitions are not associated with any types and values, i. e., tokens are indistinguishable from each other. This can lead to problems if several process instances are represented in the same Petri net. Also, representing data associated with a process instance, such as information about a customer applying for a credit, is not possible using traditional Petri nets.

Colored Petri nets (CPNs) [11] provide a means to solve these problems. In colored Petri nets tokens are associated with data types, called *color sets*, and they can take on values (or *colors*) of that color set. Hence, the enablement of a transition does not only depend on the presence of tokens in the transition's input places but also on the values of those tokens. To specify the values that are required for the enablement of a transition, a CPN also defines arc expressions for arcs and guard expressions for transitions. Arc and guard expressions are associated with variables that have a certain type (or color set) and that can be bound to the values (or colors) of the tokens in the input places. In this way, only tokens with certain values can enable and hence be consumed by a transition. Similarly, also the outgoing arcs of a transition are associated with arc expressions, such that the values of the tokens produced by the transition can be determined.

In the following, the concept of a *multiset* is used. They are defined as follows (based on [36]):

Definition 5 (Multiset). A multiset is the generalization of a set in which elements can appear more than once. Given a non-empty set S , a multiset over S is a function $m : S \rightarrow \mathbb{N}_0$ mapping each element $s \in S$ into a natural number including 0. Then, $\forall s \in S : m(s) \in \mathbb{N}_0$ denotes the multiplicity of s in m . Note that S can be interpreted as a multiset as well, where each element's multiplicity is equal to 1. \diamond

Multisets can be written in various ways. For example, given the set $\{a, b, c\}$, a possible multiset is given by $m = \{(a, 1), (b, 2), (c, 0)\}$, corresponding to the relational definition of the function m . m can also be written as $\{a^1, b^2, c^0\}$, where the superscript of each element denotes its multiplicity, or as $\{a, b, b\}$.

The usual set operations as well as addition can be defined for multisets:

Definition 6 (Operations on multisets). Let m_1 and m_2 be multisets over the set S .

- $m_1 \subseteq m_2 \iff \forall s \in S : m_1(s) \leq m_2(s)$,
- $m_1 \cap m_2 = \{(s, \min(m_1(s), m_2(s))) \mid s \in S\}$,
- $m_1 \cup m_2 = \{(s, \max(m_1(s), m_2(s))) \mid s \in S\}$,
- $m_1 \setminus m_2 = \{(s, \max(0, m_1(s) - m_2(s))) \mid s \in S\}$,
- $m_1 + m_2 = \{(s, m_1(s) + m_2(s)) \mid s \in S\}$.

Note that since sets can be interpreted as multisets, these operations can also be applied to two sets, or on a set and a multiset. \diamond

Finally, given a set S , the set of *all* possible multisets over S is denoted S_{MS} . Based on [36], colored Petri nets are then formally defined as follows:

Definition 7 (Colored Petri net). A colored Petri net is a nine-tuple $CPN = (P, T, A, \Sigma, V, col, grd, ex, init)$ where

- P is a finite set of places,
- T is a finite set of transitions, with $T \cap P = \emptyset$,
- $A \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs,
- Σ is a finite set of non-empty color sets,
- V is a finite set of typed variables such that the type of each $v \in V$ is given by some color set in Σ ,
- $col : P \rightarrow \Sigma$ is a color set function assigning a color set to each place,

- $grd : T \rightarrow Expr$ is a guard function assigning a guard expression to each transition $t \in T$, such that the codomain of $grd(t)$ is the Boolean values,
- $ex : A \rightarrow Expr$ is an arc expression function assigning an arc expression to each arc $a \in A$, such that the codomain of $ex(a)$ is equal to $col(p)_{MS}$, where $p \in P$ is the place connected to a , and
- $init : P \rightarrow Expr$ is an initialization function that assigns a multiset of colors to each place $p \in P$, such that the codomain of $init(p)$ is equal to $col(p)_{MS}$.

◇

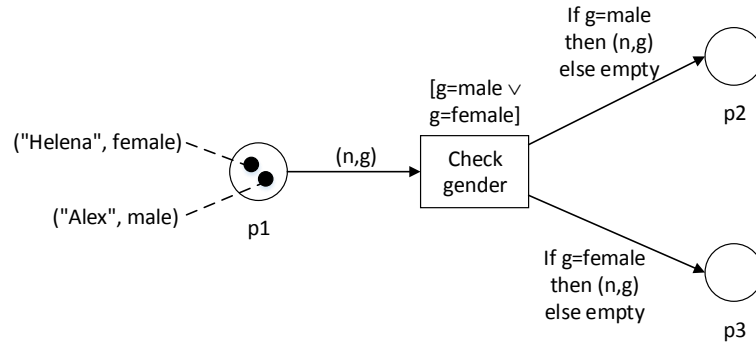


Figure 2.4: A simple colored Petri net checking the gender of a person

Figure 2.4 shows a very simple colored Petri net that checks the gender of a person. In this example:

- $\Sigma = \{Person, Gender, String\}$, where $Person = String \times Gender$ and $Gender = \{male, female\}$,
- $V = \{n : String, g : Gender\}$,
- $\forall p \in P : col(p) = Person$,
- $\forall t \in T : grd(t) = g=male \vee g=female$,

$$ex(a) = \begin{cases} (n, g) & \text{if } a = (p1, \text{Check gender}) \\ \text{if } g=male \\ \text{then } (n, g) \\ \text{else empty} & \text{if } a = (\text{Check gender}, p2) \\ \text{if } g=female \\ \text{then } (n, g) \\ \text{else empty} & \text{if } a = (\text{Check gender}, p3) \end{cases}$$

$$\bullet \text{ init}(p) = \begin{cases} \{("Helena", female), ("Alex", male)\} & \text{if } p = p1 \\ \emptyset & \text{otherwise} \end{cases}$$

To define the enablement and firing of a transition, additional concepts are required, which are the following (based on [36]):

Definition 8 (CPN concepts). Let $CPN = (P, T, A, \Sigma, col, grd, ex, init)$ be a colored Petri net.

- M is a marking function assigning a multiset of tokens to each place $p \in P$, where each token is of type $col(p)$. Therefore, $\forall p \in P : M(p) \in col(p)_{MS}$,
- The initial marking M_I is given by the initialization function $init(p)$, for all $p \in P$,
- for all $t \in T$, the variables of transition t are given by $Var(t) \subseteq V$, consisting of the variables of the guard of t and the variables of the arc expressions of the arcs connected to t ,
- a *binding* of a transition t is a function b assigning a value $b(v)$ to each $v \in Var(t)$, where $b(v)$ is in the domain of v . The set of all bindings for a transition t is given by $B(t)$,
- a *binding element* is a pair (t, b) , where $t \in T$ and $b \in B(t)$. The set of all binding elements $BE(t)$ for a transition t is defined by $BE(t) = \{(t, b) \mid b \in B(t)\}$. The set of all binding elements in a CPN is denoted BE .

◇

For a given binding element (t, b) , $grd(t)(b)$ denotes the result of evaluating $grd(t)$ for the binding b , yielding either *true* or *false*. Moreover, $ex(a)(b)$ is the result of evaluating $ex(a)$ for the binding b , yielding the tokens required to be on place p to satisfy the arc expression of a under binding b . This leads to the definition of the firing rule of a CPN transition [36]:

Definition 9 (CPN transition firing rule). Let $CPN = (P, T, A, \Sigma, col, grd, ex, init)$ be a colored Petri net and M a marking. A transition $t \in T$ is enabled if there exists a binding b such that the binding element (t, b) is enabled in a marking M . This is the case if and only if:

- (i) $grd(t)(b) = true$, and
- (ii) $\forall p \in \bullet t : ex((p, t))(b) \subseteq M(p)$.

◇

Given that the binding element (t, b) is enabled in marking M , transition t can fire, leading to marking M' , denoted by $M \xrightarrow{(t,b)} M'$. The new marking M' is defined as follows:

$$\forall p \in P : M'(p) = (M(p) \setminus ex((p,t))(b)) + ex((t,p))(b).$$

Therefore, the tokens required by the expressions of the incoming arcs of the firing transition are consumed from the transition's input places, and the tokens specified by the expressions on the outgoing arcs of the firing transition are produced on the transition's output places.

Similar to the reachability graph of Petri nets defined in Definition 3, the reachability graph of a colored Petri net is defined as follows (based on [36]):

Definition 10 (CPN reachability graph). Let $CPN = (P, T, A, \Sigma, col, grd, ex, init)$ be a colored Petri net with binding elements BE , and M a marking. A directed graph $G = (V, E, l)$ is the reachability graph of CPN if:

- $V = \{M \mid M_I \xrightarrow{*} M\}$ is the set of vertices corresponding to the reachable markings of the colored Petri net,
- $E = \{(M, (t, b), M') \in V \times BE \times V \mid M \xrightarrow{(t,b)} M'\}$, is the set of edges such that there is an edge for each binding element leading from one marking to another,
- $l : E \rightarrow BE, e \mapsto e^i$, where $e^i, i \in \mathbb{N}_{>0}$ refers to the i th element of the tuple e , is the labeling function assigning a label to each edge corresponding to the binding element of that edge.

◇

2.1.2.3 Business Process Model and Notation

The Business Process Model and Notation (BPMN) standard is a specification published by the Object Management Group (OMG)¹. Its latest version was released in December 2013 as BPMN 2.0.2 [59]. BPMN is the de facto industry standard for modeling business processes [67]. It consists of notational elements as well as a meta model that describes how these elements can be combined to design business process models. Moreover, additional non-graphical attributes can be defined for such models, specifying additional information up to the point of making the model executable. These models can be of different types, e. g., describing the orchestration of single business processes or the collaboration of processes of multiple organizations. In this thesis, we focus on so called business process diagrams modeling the process of a single organization.

¹ <https://www.omg.org>

Business process diagrams can be made up of numerous different notational elements than can be categorized according to Figure 2.5. This figure shows three different categories: flow objects, connecting

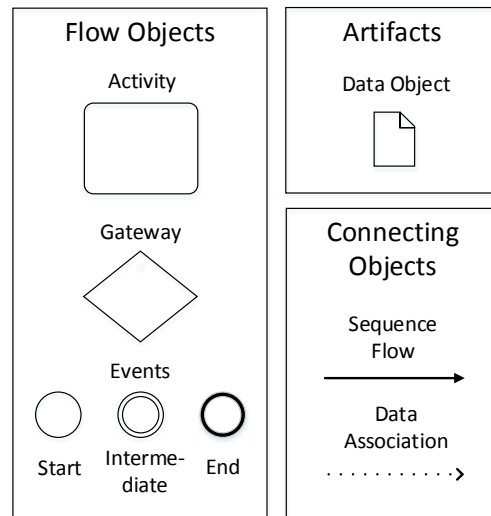


Figure 2.5: BPMN notational elements for business process diagrams considered in this thesis

objects and artifacts. Flow objects are nodes that can be part of a sequence flow. Thereby, they are the elements that drive the process execution from start to finish.

The activity is the most important flow object of BPMN [32]. It represents an item of work whose execution requires a certain amount of time and is represented by a rectangle with rounded corners. Activities can be atomic or decomposable. Atomic activities are called tasks. Decomposable activities represent sub-processes and thus encapsulate another process that is again made up of the elements in Figure 2.5. As already mentioned, activities are connected via sequence flows that describe causal dependencies between the activities. Therefore, activities and sequence flows already satisfy the basic definition of a business process given in the beginning of Section 2.1 stating that a business process “consists of a set of activities that are performed in coordination”.

Tasks (or atomic activities) can have different *task types*, represented by a corresponding symbol in the upper left corner of the task rectangle. Prominent examples are *user tasks* that are executed by persons interacting with a software system, *service tasks* that are executed automatically by a web service or some other automated application, *send* and *receive tasks* for sending and receiving messages, and *business rule tasks*, that provide input to some sort of decision system and produce a corresponding decision output. In this thesis, the *business rule task*

will be of particular importance, and will henceforth be called *decision task*.

Usually, a business process does not only consist of a linear execution of activities. In many cases, alternative paths of execution can be chosen or a set of paths can be followed concurrently. Such behavior is realized using a gateway, represented by a diamond symbol. Inside the diamond, a marker specifies the behavior of the gateway. A gateway that is followed by alternative execution paths contains an x and is called exclusive gateway, and a gateway that is followed by one or more execution paths contains an o and is called inclusive gateway. To specify which execution path(s) should be chosen in a given situation, each outgoing flow of the gateway is annotated with a condition that evaluates to true or false. A gateway that is succeeded by concurrent execution paths contains a $+$ and is called parallel gateway. Moreover, gateways can in general be split or join gateways. Split gateways have exactly one incoming sequence flow and at least two outgoing sequence flows, whereas join gateways have at least two incoming sequence flows and exactly one outgoing sequence flow.

The third type of flow objects in Figure 2.5 are the events. Unlike activities that need time to be carried out, events simply occur and as a result do not take time. They essentially enable the process to communicate with its environment. Hence, there are two categories of events, catching and throwing. A catching event enables the process to receive events (or information) from its environment, while a throwing event expresses that the process supplies information to its environment.

Along a different dimension, events can further be distinguished into *start*, *intermediate*, and *end* events as shown in Figure 2.5. Processes are triggered by start events from the environment. Therefore, they are always of catching nature. Intermediate events can occur during process execution. They are either caught or thrown by the process. Finally, the process terminates with an end event. Therefore, end events are always of throwing nature. Events can be of a particular *type*. For example, a catching event that represents the reception of a message contains a letter symbol. For more information, the reader is referred to the BPMN standard [59].

The next category of elements in Figure 2.5 are artifacts. Since the execution of activities in business processes usually consume and produce data, the most important artifact is the *data object*. Data objects can be connected to an activity to represent that this activity reads and/or writes such an object. Moreover, catching events can write data objects, and throwing events can read data objects. In BPMN the data object will simply be identified by a name such as *Order* and can also have a state that changes after the object was read and written. The exact definition and structure of the data object, however, is outside of the scope of BPMN. For this purpose, other modeling

languages such as the Unified Modeling Language (UML) can be employed. At the same time, we assume that such a definition exists, so that the process model can refer to the various attributes of the data object, e. g., in the conditions that the outgoing flows of an exclusive/inclusive gateway are annotated with.

Finally, the *connecting objects* connect flow objects and artifacts with each other. The *sequence flow* connects all flow objects with each other, while the *data association* connects activities with data objects, where the direction of the edge indicates whether that data object is read or written.

The following definition describes the above mentioned elements and how they are related more formally:

Definition 11 (Process model). A process model m is a tuple $(N, DO, C, CF, DF, \alpha, \xi, \delta)$ where:

- $N = A \cup G \cup E$ is a finite non-empty set of flow objects, with set A of activities, set G of gateways, set E of events, and set $A_D \subseteq A$ of decision tasks,
- $E = CT \cup TH$, where CT is the set of catching events, TH the set of throwing events, and $e_s \in CT$ the process' (only) start event,
- DO is a finite set of data object nodes,
- C is a finite set of conditions. Each condition $cond \in C$ is a Boolean expression over an attribute $attr$ of some data object in DO , such that $cond(attr) \in \{true, false\}$,
- $CF \subseteq N \times N$ is the control flow relation such that each edge connects two control flow nodes,
- $DF \subseteq (DO \times (A \cup TH)) \cup ((A \cup CT) \times DO)$ is the data flow relation indicating read or write operations of an activity or event with respect to a data node,
- function $\alpha : G \rightarrow \{and, xor, ior\}$ assigns to each gateway a type in terms of a control flow construct, where *and* represents the parallel, *xor* the exclusive and *ior* the inclusive gateway,
- $\xi : (G \times N) \cap CF \rightarrow C$ is a function that assigns exclusive (i. e., non-overlapping) conditions to control flow edges originating from gateways of type *xor* and *ior* with multiple outgoing edges, and
- $\delta : A_D \rightarrow DM$ is a function assigning a decision model to each decision task. Decision models are defined in Definition 19.

◇

For any node $n \in N$, the predecessors of n are denoted by $\bullet n = \{n' \mid (n', n) \in CF\}$, whereas its successors are given by $n\bullet = \{n' \mid (n, n') \in CF'\}$.

Definition 12 (Process fragment). Let $m = (N, DO, C, CF, DF, \alpha, \zeta, \delta)$ be a process model. A *process fragment* $pf = (N', DO', C', CF', DF', \gamma, \beta, \sigma)$ is a connected subgraph of process model m such that $N' = A' \cup G' \cup E' \subseteq N$, $A' \subseteq A$, $A'_D \subseteq A_D$, $G' \subseteq G$, $E' \subseteq E$, $DO' \subseteq DO$, $C' \subseteq C$, $CF' \subseteq CF$, and $DF' \subseteq DF$. Functions γ , β and σ are restrictions of functions α , ζ , and δ respectively with corresponding new domains. \diamond

2.1.2.4 Running Example

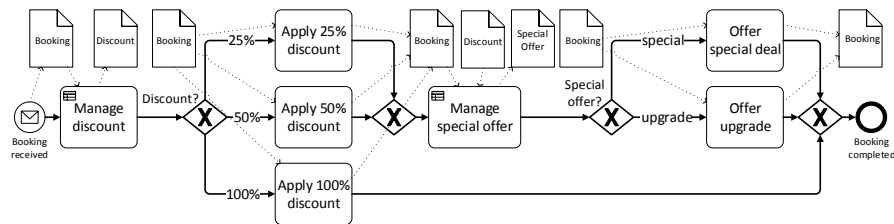


Figure 2.6: Train ticket booking process derived from booking tickets with Deutsche Bahn, modeled as a BPMN process diagram

Figure 2.6 shows a business process diagram consisting of most of the elements described in the previous section. The process was derived from booking train tickets with Deutsche Bahn, a German railway company², and handles the discounting of tickets for customers that own a discount card. The process begins when the customer submitted their booking request. The contents of this message are then written into a *Booking* data object. The internal structure of this object is detailed in a UML class diagram depicted in Figure 2.7.

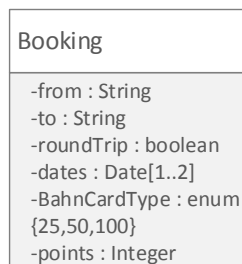


Figure 2.7: UML class diagram for the *Booking* data object in Figure 2.6

On the one hand, the booking object contains information about the trip, such as start and destination, an indication whether or not it is a round trip, and corresponding dates. On the other hand, it holds data that is important for the discounting. Customers can own a discount card, called *BahnCard*, which exists in three different types: 25,

² <https://www.bahn.com/en/view/index.shtml>

50, and 100. Each of these types correspond to a respective discount in percent. For example, a BahnCard of type 25 provides the customer with a discount of 25% on ticket prices. Moreover, customers can collect *points* on this card by purchasing tickets.

Thus, the *Manage discount* task reads the *Booking* data object and based on that information writes a *Discount* data object. Note that this task is a decision task, meaning that some decision system is invoked to take the decision. The details of this will be described in Section 2.3. The value of the discount will then be used by the succeeding gateway to route the process in the appropriate direction, such that a discount can be applied to the booking. Given that the discount does not equal 100%, another decision is invoked to determine a potential deal or upgrade. After this has been handled, the booking is completed.

2.1.3 Verification of Business Process Models

An important step in the lifecycle in Figure 2.1 is the analysis of the (re)designed process models. This step can roughly be divided into verification and validation. Verification is an important act to make sure that the process model is free of errors. This can include different aspects of the model. For example, one may consider only the control flow or also include the handling of data objects. The most important property in this regard is soundness, defined by van der Aalst in 1998 [16], but also compliance checking has become a topic of interest [31].

Soundness is defined for workflow nets (cf. Definition 4) and also compliance checking requires a formal model, so that a business process model that is modeled in BPMN first needs to be translated to a workflow net (or Petri net) to be able to perform the verification. This translation will be described in the following, while the section thereafter explains how to check the soundness of a workflow net.

2.1.3.1 Mapping BPMN Models to Petri Nets

Dijkman et al. propose a mapping from BPMN to Petri nets that can be employed for this purpose [30]. Figure 2.8 shows an excerpt of the mapping rules, which are required for the elements in Figure 2.5.

The left part of the figure shows how to translate BPMN events and activities to Petri net fragments, whereas the right part illustrates the mapping for parallel and exclusive gateways. A start event s that is followed by a flow object y is mapped to the initial place in the Petri net connected to a transition. This transition fires when the start event occurs and is therefore also labeled s . Transition s is then connected to the input place of the succeeding flow object y .

An intermediate event e (second row) is mapped similarly to a transition e which is connected to the input and output places of the preceding and succeeding objects respectively. Note that in many cases

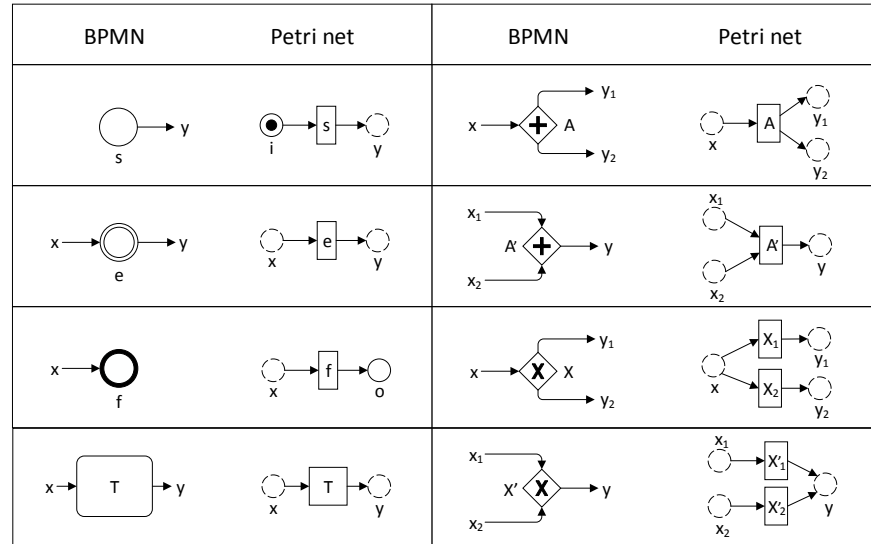


Figure 2.8: Mapping of BPMN task, events and gateways to Petri nets (cf. [30])

the output place of the preceding object can be merged with the input place of the succeeding object. For example, in a process model in which a start event is followed by some intermediate event, the output place y of the start transition is merged with the input place x of the intermediate event transition.

An end event f is mapped— analogously to the start event—to a transition f that receives a token from its preceding object x and upon firing produces a token on the final place o . Note that the initial and final places cannot be merged with any other places and are therefore represented with a solid line instead of a dashed line.

An activity T in BPMN is mapped in the same way as an intermediate event consuming a token from its input place and producing a token on its output place.

The mapping of gateways, illustrated in the right part of Figure 2.8, nicely illustrates their semantics, especially the difference between split and join gateways. A parallel split gateway A (first row) is translated to a corresponding transition in the Petri net. As mentioned in Section 2.1 this gateway has exactly one incoming edge and at least two outgoing edges. Hence, the Petri net translation consumes a token from its preceding object and produces a token for each input place of its succeeding objects. On the contrary, a parallel join gateway has at least two incoming edges and exactly one outgoing edge. Therefore, it must consume a token from each of its preceding objects in order to fire and produce a token for its succeeding object.

An exclusive split gateway has the same connection constraints as the parallel split gateway. However, it will take a decision. It will only enable exactly one of its succeeding objects. This behavior is

represented in the Petri net by connecting the input place x with as many transitions X_1, X_2, \dots, X_n as there are outgoing edges of gateway X . Hence, exactly one of these transitions will be able to consume a token from place x , such that only for the output place of that transition a token will be produced. In contrast, an exclusive join gateway X' (last row) signals its outgoing edge each time one of its incoming edges is signaled. Accordingly, for each incoming edge a transition X'_i is created and each X'_i can independently produce a token on the only output place y .

Note that Figure 2.8 does not mention how to map inclusive gateways, although they are allowed in Definition 11. Such gateways must therefore first be mapped to a combination of exclusive and parallel gateways, as discussed in [34, 51].

2.1.3.2 Soundness Verification

Having translated a business process model to a workflow net one can verify its soundness. There exist various notions of soundness that differ with respect to the requirements they pose on the investigated workflow net. These notions have been summarized in [44]. In the following, each notion is defined and exemplary workflow nets are shown.

All notions of soundness are based on the original *classical soundness* that has three (seemingly) simple requirements [16]:

- (i) Each process instance eventually terminates;
- (ii) when an instance terminates, there is exactly one token in the net and it is in the final place;
- (iii) each task can participate in at least one process instance.

More formally, classical soundness of a workflow net is defined as follows (cf. [53]):

Definition 13 (Classical soundness). Let $N = (P, T, F)$ be a workflow net with initial place $i \in P$ and final place $o \in P$, and M a marking. N is sound if and only if:

- (i) $\forall M : ([i] \xrightarrow{*} M) \implies (M \xrightarrow{*} [o]),$
- (ii) $\forall M : ([i] \xrightarrow{*} M \wedge M \geq [o]) \implies (M = [o]),$
- (iii) $\forall t \in T, \exists M, M' : [i] \xrightarrow{*} M \xrightarrow{t} M'.$

◇

Consider the workflow net in Figure 2.9, which is the same as in Figure 2.2, repeated for convenience. This workflow net is not sound because after firing transition $t1$ and $t4$ the net's marking will be $[p2, p4]$. Hence, condition (ii) of Definition 13 will be violated because

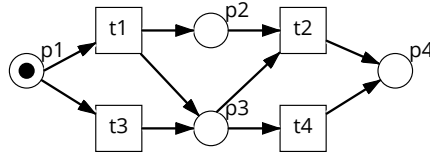


Figure 2.9: A workflow net that is not sound, but relaxed sound

it is a marking in which there is a token in the final place (p_4) but also in another place.

The workflow net in Figure 2.9, however, is *relaxed sound*. Relaxed soundness allows unsound firing sequences to occur. Unsound firing sequences violate conditions (i) or (ii) in Definition 13. However, every transition must be able to participate in at least one sound firing sequence [19].

Definition 14 (Relaxed soundness). Let $N = (P, T, F)$ be a workflow net with initial place $i \in P$ and final place $o \in P$, and M a marking. N is relaxed sound if and only if:

$$(i) \quad \forall t \in T, \exists M, M' : [i] \xrightarrow{*} M \xrightarrow{t} M' \xrightarrow{*} [o].$$

◇

As mentioned before, firing t_1 and t_4 will not lead to a sound firing sequence. Still, t_1 can be part of a sound firing sequence by firing t_2 after t_1 . Similarly, firing t_3 before t_4 will lead to a sound sequence in which t_4 is involved.

Another notion that is slightly less restrictive than classical soundness is that of *weak soundness*. This notion allows dead transitions, which are transitions that cannot participate in any process instance, thereby violating condition (iii) in Definition 13 [24].

Definition 15 (Weak soundness). Let $N = (P, T, F)$ be a workflow net with initial place $i \in P$ and final place $o \in P$, and M a marking. N is weak sound if and only if:

$$(i) \quad \forall M : ([i] \xrightarrow{*} M) \implies (M \xrightarrow{*} [o]),$$

$$(ii) \quad \forall M : ([i] \xrightarrow{*} M \wedge M \geq [o]) \implies (M = [o]).$$

◇

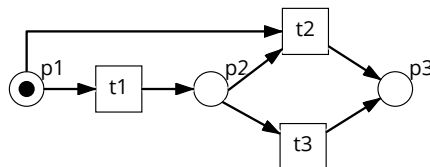


Figure 2.10: A weak sound workflow net

Figure 2.10 shows a workflow net that is neither classically sound nor relaxed sound, because of the dead transition $t2$. Since the net without $t2$ fulfills all conditions (i)–(iii) in Definition 13, however, the net is weak sound.

Even more permissive than weak soundness is the notion of *lazy soundness*. In addition to the possible soundness violations of weak soundness, lazy soundness allows the net to be lazy in the sense that there can be tokens left in the net after a token appeared on the final place. Still, the remaining tokens are not allowed to appear on the final place later on, i. e., they have to remain somewhere else in the net [25]. Therefore, in addition to allowing violations of condition (iii) in Definition 13, conditions (i) and (ii) are weakened:

Definition 16 (Lazy soundness). Let $N = (P, T, F)$ be a workflow net with initial place $i \in P$ and final place $o \in P$, and M a marking. N is lazy sound if and only if:

- (i) $\forall M, \exists M' : ([i] \xrightarrow{*} M) \implies (M \xrightarrow{*} M' \wedge M'(o) = 1)$,
- (ii) $\forall M : ([i] \xrightarrow{*} M) \implies (M(o) \leq 1)$.

◇

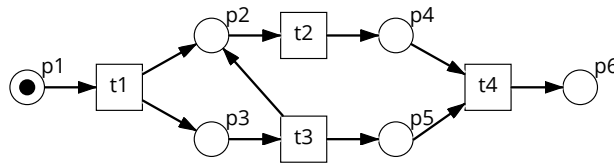


Figure 2.11: A workflow net that is neither sound, nor weak sound, nor relaxed sound, but lazy sound

Figure 2.11 shows a workflow net that is neither sound, nor weak sound, nor relaxed sound. This is because this net does not allow a single sound firing sequence. There will always be a token left on place $p4$. However, every firing sequence will lead to a token on place $p6$ and the remaining token on place $p4$ will always stay there. Therefore, the net is lazy sound.

Finally, the notion of *easy soundness* is more permissive than both weak and relaxed soundness. Its only simple requirement is that it contains a single sound firing sequence [23]. Therefore, it is defined as follows:

Definition 17 (Easy soundness). Let $N = (P, T, F)$ be a workflow net with initial place $i \in P$ and final place $o \in P$, and M a marking. N is easy sound if and only if:

- (i) $\exists M : ([i] \xrightarrow{*} M) \implies (M \xrightarrow{*} [o])$.

◇

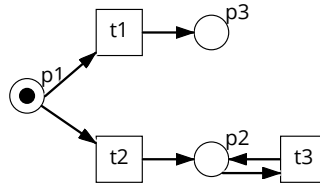


Figure 2.12: A workflow net that is only easy sound

Figure 2.12 illustrates a workflow net that has one sound firing sequence and another firing sequence that will lead to a livelock. Hence, it is only easy sound.

Besides the soundness notions discussed above, [44] additionally considers *generalized*, *up-to- k* and *k -soundness*. All of these notions analyze the behavior of workflow nets when there are multiple tokens in the initial place i . Yet, this thesis only considers workflow nets that are the result of translating business process models to Petri nets using the mapping rules displayed in Figure 2.8. Therefore, the initial place will only contain a single token, such that the remaining soundness notions are irrelevant.

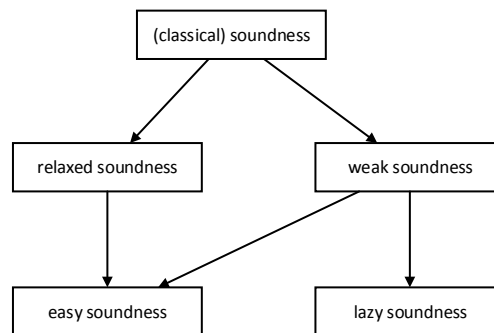


Figure 2.13: Various notions of soundness and their relationships [44]

Figure 2.13 gives an overview of the different notions of soundness defined above and also shows their relationships, where an arrow from source to target can be interpreted as an implication, giving rise to a tree structure. For example, classical soundness represents the root node of that tree because it is the most restrictive notion. It implies relaxed and weak soundness, i. e., every workflow net that is classically sound is also relaxed and weak sound. Regarding weak soundness this implication is obvious because it simply omits condition (iii) of classical soundness. Relaxed soundness in a way combines conditions (i) and (iii) of Definition 13 because it requires that every transition can fire in some marking and that through the following marking the final state can be reached. However, it does not require that from every marking the final state can be reached, e. g., deadlocks are allowed.

Weak soundness in turn implies lazy soundness. Both conditions of lazy soundness are more permissive than the conditions of weak soundness because they only require that from every marking a marking can be reached that contains a token in the final place, and that the final place will never contain more than one token. Weak soundness on the other hand requires a token on the final place and nowhere else.

Lazy soundness, however, is not implied by relaxed soundness because the latter does not require that from *every* marking a marking with a token in the final place can be reached. Of course, there is also no implication in the other direction because relaxed soundness requires that every transition is part of a sound firing sequence, which is not required by lazy soundness. Both, weak and relaxed soundness do imply easy soundness because the latter only requires at least one sound firing sequence, which is guaranteed by the former two (but not by lazy soundness).

For a given workflow net the various soundness notions defined above can be easily checked on the basis of the reachability graph introduced in Section 2.1.2.1. Figure 2.14 shows the reachability graph corresponding to the Petri net in Figure 2.9.

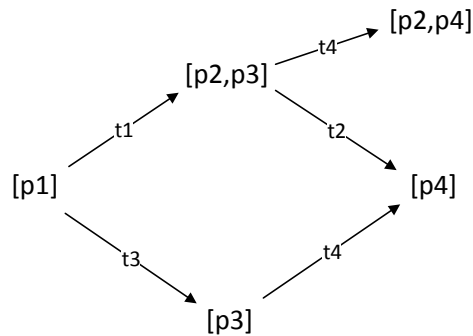


Figure 2.14: Reachability graph for the Petri net shown in Figure 2.9

For example, it is obvious that the net is neither classically sound nor weak sound, since not from every marking there is a continuation to the final marking $[p4]$, thereby violating condition (i). However, the net is relaxed sound because every transition $t1-t4$ is part of a firing sequence in Figure 2.14 leading to the final marking. Hence, by implication, the net is also easy sound. In addition, the net is also lazy sound as every transition sequence will lead to a marking that contains exactly one token in the final place.

2.2 BUSINESS DECISION MANAGEMENT

Business process modeling is an effective means for the documentation and implementation of business operations. Moreover, the frequent use of the exclusive split gateway in process models [32] shows

that there are also many decisions that need to be taken during process execution. However, although this gateway is only intended to route the process flow based on the final result of some decision-making procedure, it is often misused to represent the decision logic itself. This results in a sequence of gateways with many branches, leading to incomprehensible models that are hard to maintain. This issue will be discussed in more detail in Chapter 3.

To adequately model the decisions that need to be taken in the context of a business process, the OMG developed the Decision Model and Notation (DMN) standard. Its first version was published in 2015 [61] and version 1.1 shortly after in 2016 [76]. Moreover, version 1.2 was released in January 2019 [114]. This demonstrates the strong interest in this standard, leading to feedback in the form of feature requests as well as based on the analysis of DMN models designed since the standard's release.

2.2.1 DMN Decision Models

Decision models in DMN consist of two artifacts. As a high-level overview, *decision requirements diagrams* (DRDs) specify the inputs that are necessary to make the decision. Such inputs can be actual input data that may come from a process associated with the decision. But also decisions itself may be used as inputs to other decisions. This gives rise to a tree-like structure for DRDs. Figure 2.15 gives an example of an abstract DRD. It consists of two *decisions* represented as rectangles and two *input data* elements, represented as ellipses. The arrows between those elements are called *information requirement edges* since they carry information from their source to their target.

Decision 1 is the top-level decision of this decision model. Therefore, its output will be returned to the caller of the decision model. The other decisions are called sub-decisions whose outputs are used as inputs for the top-level decision or other sub-decisions. Analogous to the fact that the output of the top-level decision will be returned to the caller, the caller must also provide the values for the input data elements. These values are then used for the various decisions in the DRD.

Besides decisions and input data, DRDs may contain two other types of nodes: *business knowledge models* and *knowledge sources*. Business knowledge models are similar to decision elements, since they also encapsulate decision logic. However, the focus here is on reusable business know-how of the organization that can be applied to multiple decision models. Knowledge sources, on the other hand, are solely used for documentation purposes. Hence, they can be connected to decision nodes to explain what the decision logic is based on, such as a legal requirement or an organizational policy. Still, this thesis will only consider decision and input data nodes, since these are the main

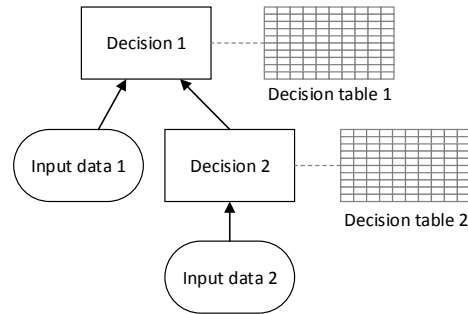


Figure 2.15: An abstract DMN decision requirements diagram

ingredients of a DRD while the other elements do not provide any additions relevant for the considerations in this thesis.

In order for a decision to transform the inputs it receives to an output, it needs to be associated with some decision logic. Therefore, the other artifact of DMN, besides DRDs, is the *decision logic level*. On this level, decisions can be made executable by specifying the details of how inputs are mapped to outputs. Decision logic can be represented in many ways, e. g., by plain text, a computer program, an analytical model, or a decision table. Decision tables are standardized in DMN. They are easy to create, understand and change, and are suited for both business users who have knowledge about the business logic and software developers that implement the decision models.

F	input1 <i>Number</i>	input2 <i>{a,b}</i>	output <i>{w,x,y,z}</i>
1	1	a	w
2	[1..2]	a	x
3	< 1	b	y
4	> 2	-	z

Figure 2.16: An example of a DMN decision table

Figure 2.16 shows an example of a DMN decision table. By default, tables consist of rows corresponding to rules and columns corresponding to inputs or outputs. Each input/output column is associated with a data type, such as string or integer, and can be optionally restricted to a subset of possible values from that domain, which is displayed directly below the input's/output's name. A decision table then describes a relation between the set of input values and the set of output values. In this regard, the rules of the table play the most important role. A rule specifies a logical expression for each input (i. e., a condition), that tests the input for a certain value or range of values. The result of that test will be either true or false. Additionally, a rule can declare a certain input as irrelevant using a dash symbol ("–"). This means that any value for that input will lead to true. At runtime, given a value for each input, a rule is said to match if the

conjunction of all logical expressions of that rule evaluates to true, and the corresponding output value can be determined. Hence, a decision table is considered *complete* if for all possible input values at least one rule matches.

However, it may be the case that for some input values more than one rule matches. This happens when the table contains overlapping rules. Two rules are overlapping if all of their corresponding input conditions overlap, i. e., they share a common value or range of values. Since these overlapping rules do not necessarily map to the same output value, a DMN table can define a *hit policy* that determines how the final output value should be determined.

Consider, for example, rules 1 and 2 of the table in Figure 2.16. Both of these rules match for the input $input1 = 1$ and $input2 = a$, or $(1, a)$ for short. Yet, they lead to different outputs, w and x respectively. Therefore, the table specifies a *first* hit policy, indicated by the letter F in the table's upper left corner. This policy dictates that if multiple rules match for an input, the rule that comes first in the decision table is chosen to return the output. Consequently, since rule 1 comes first in the example table, the input $(1, a)$ will lead to the output w . In the following, we will describe the various DMN hit policies in detail.

2.2.2 DMN Decision Table Hit Policies

As already mentioned above, DMN decision tables are allowed to have overlapping rules. This leads to conflicts when a given input matches more than one rule. Such conflicts are settled based on a variety of hit policies. In principle, DMN divides these policies into *single-* and *multi-hit* policies. For a given input, the former always returns the output of a single rule—even if multiple rules match—while the latter returns the outputs of all matching rules, potentially aggregated in some way.

There are four single-hit policies, only one of which guarantees that only one rule matches for a given input:

- *unique*: all rules of the table are exclusive, such that at most one rule matches for a given input.
- *any*: multiple rules can match for a given input, but they all have the same output such that there are no ambiguities.
- *first*: if multiple rules match for a given input, the rule that comes first in the decision table is chosen to return the output.
- *priority*: similar to first-hit, but in this case the rule with the highest priority is chosen, where the priority is given by the ordered set of allowable values displayed below the output's name (cf. Figure 2.16).

The multi-hit policies are specified as follows:

- *rule order*: all matching rules' outputs are returned in a list ordered by the appearance of the rules in the table.
- *output order*: all matching rules' outputs are returned in a list ordered by the priority of the rules.
- *collect*: the outputs of all matching rules are collected and then aggregated in a predefined way, for example by summing up the output values or taking the maximum.

In this thesis, all of those hit policies are covered.

2.2.3 Formalization of DMN Decision Models

In this section, formal definitions of decision tables and decision requirements diagrams are given. The following definition of a DMN decision table is based on the definitions given in [12] and [97].

Definition 18 (DMN decision table). A decision table dt is a tuple $(I, O, dom, R, prio, p, c)$ where:

- $I = \{i_1, i_2, \dots, i_n\}$ is a set of $n > 0$ input variables,
- $\forall i \in I, dom(i)$ is the domain of input variable i , where dom is a function mapping a variable to its domain,
- o is an output variable with domain $dom(o)$,
- given the input value combinations $IV = \prod_{j=1}^n dom(i_j)$ and the set of output values $OV = dom(o)$, the finite set of $q > 0$ rules is defined as $R = \{r_1, r_2, \dots, r_q\}$, where $|R| = q, q \in \mathbb{N}$ and $\forall r \in R : r \subseteq IV \times OV$,
- $prio : R \rightarrow \{1, \dots, |R|\}$ assigns each rule a priority visualized by its rule number in the leftmost column of the table. If no priority is explicitly given, it is implicitly given by the graphical ordering of the rules in the table,
- $p \in \{unique, any, priority, first, output\ order, rule\ order, collect\}$ indicates the decision table's hit policy,
- $c \in Bool$ indicates whether or not the decision table is complete.

◇

Example. Consider the decision table depicted in Figure 2.16. This table can be formally defined as follows.

- $I = \{input_1, input_2\}, o = output$,
- $dom(input_1) = \mathbb{R}, dom(input_2) = \{a, b\},$
 $dom(output) = \{w, x, y, z\},$

- $R = \{r_1, r_2, r_3, r_4\}$, where $\forall r \in R : r \subseteq (\mathbb{N} \times \{a, b\}) \times \{w, x, y, z\}$. For example, $r_4 = (\{n \mid n \in \mathbb{N} \wedge n > 2\} \times \{a, b\}) \times \{z\}$, because the expression “–” allows any value for the variable *input2*,
- $prio = \{(r_1, 1), (r_2, 2), (r_3, 3), (r_4, 4)\}$,
- $p = first$ (denoted by F in the upper left corner),
- $c = false$. For example, no rule matches for the input $(0, a)$.

Decision models made up of a decision requirements diagram and decision logic based on decision tables can then be defined as follows.

Definition 19 (Decision model). A decision model dm is a tuple (D, InD, IR, tab) where:

- D is the set of decision nodes,
- $d_{top} \in D$ is the top-level decision, determining the output of dm ,
- InD is the set of input data nodes,
- $IR \subseteq \{D, InD\} \times D$ is the set of directed information requirement edges,
- $tab : D \rightarrow DT$ assigns each decision $d \in D$ a decision table.

◇

2.2.4 Decision Table Analysis

Similar to the fact that business process models can be analyzed regarding certain correctness criteria, also decision tables have desirable properties that can be verified. For example, [8] and [9] describe potential problems of rule sets in knowledge bases and how to check those. The problems analyzed include consistency issues such as redundant and conflicting rules and completeness of the rule set. Based on these problems, [14] derives “major modelling issues of decision table construction”.

An efficient way to detect overlapping and missing rules in a DMN decision table is described in [64, 97]. This approach is based on a geometric interpretation of DMN tables. Under this interpretation, the rules of a table are represented as hyperrectangles with n dimensions, where n corresponds to the number of inputs of the table.

Consider, for example, the DMN table displayed in Figure 2.17. The geometric interpretation of that table is given in Figure 2.18 as a set of 2-dimensional hyperrectangles (i. e., rectangles).

Each rectangle corresponds to a rule of the table and is labeled with a corresponding identifier. For instance, the rectangle $r2$ represents the second rule of the table. This rule states that the credit rating will

R	Income (k) <i>Number</i> ≥ 0	Assets (k) <i>Number</i> ≥ 0	Credit Rating (A, B, C, D, E)
1	≤ 30	≤ 30	A
2	$\in [10..60]$	$\in [10..25]$	B
3	$\in [20..95]$	$\in [40..90]$	C
4	≥ 80	-	D
5	≥ 40	≥ 85	E

Figure 2.17: DMN decision table to determine a credit rating

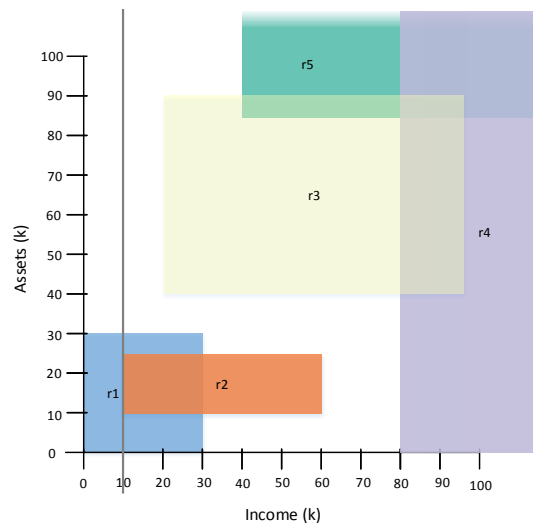


Figure 2.18: Decision table as a set of hyperrectangles corresponding to the rules in the table in Figure 2.17

be *B*, if the *Income* is between 10 and 60 and the *Assets* between 10 and 25, which coincides with the area covered by rectangle *r2*.

The problem of finding overlapping rules is now translated into the problem of finding intersecting rectangles. Similarly, finding missing rules is equivalent to locating the “white” areas that are not covered by any of the rectangles. These problems can be solved by the sweep line approach [5–7], which analyzes dimensions one by one by sweeping a line through each dimension. For example, Figure 2.18 shows a line being swept through the *Income* dimension, from left to right. The problem of finding overlapping rules, for instance, is then solved as follows: Sweep a line through the first dimension, collecting as many rules as possible that overlap in this dimension. Then, check which of the collected rules also overlap in the other dimensions, by sweeping a line through those dimensions.

2.3 INTEGRATION OF PROCESS AND DECISION MANAGEMENT

Having established a suitable way to express decisions in a dedicated model, they can be integrated with process models. The integration of process models and decision models leads to the notion of *decision-aware business process models* [35]. Such models are based on the recog-

nition that there is a substantial difference between tasks that perform work and tasks that make decisions based on data and logic, and that the details of the latter should be outsourced to decision models. The link between process models and decision models is established via *decision tasks* in the process model.

For example, the business process model in Figure 2.6 contains two decision tasks, *Manage discount* and *Manage special offer*, each linking to a decision model that makes a decision about the value that is used at the respective succeeding gateways. Such process fragments—consisting of a decision task followed by a split gateway with at least two outgoing edges where each edge is annotated with a condition corresponding to an output of the decision model associated with the preceding decision task—will be called *decision fragments*. They are defined as follows:

Definition 20 (Decision fragment). Let pf be a process fragment of a process model. Fragment pf represents a decision fragment if it starts with exactly one decision task t , that is directly followed by a split gateway g , where $\gamma(g) = xor$. g has at least two outgoing branches, and each outgoing branch of g is annotated with a condition. Moreover, t reads one or more data objects and writes exactly one data object. The data objects read by t correspond to the input data of the referenced decision model, and the conditions of the gateway refer to an attribute of the data object written by t . \diamond

An abstract decision fragment is illustrated in Figure 2.19.

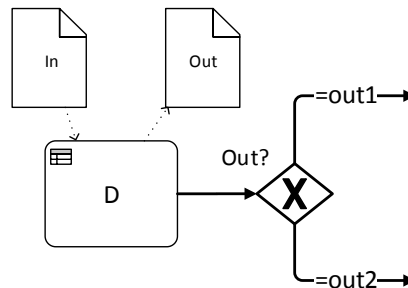


Figure 2.19: An abstract decision fragment conforming to Definition 20

The process model in Figure 2.6 includes two decision fragments. The decision task of the first is associated with the decision model displayed in Figure 2.20. Its decision requirements diagram shows that it consists of one decision that receives one input, namely the attribute *BahnCardType* of the *Booking* data object (cf. Figure 2.7). The decision is associated with a decision table that maps each possible type to a respective discount. Note that if the decision model consists of one decision only, as is the case in this example, it is sufficient to only show its decision table and omit the decision requirements diagram, since the latter does not contain any additional information.

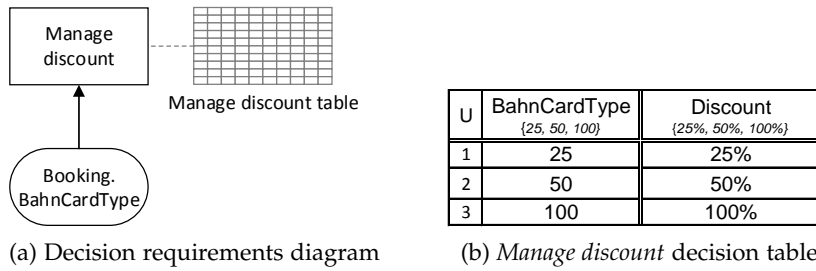


Figure 2.20: Decision model referenced by the task *Manage discount* in Figure 2.6

The other decision task in Figure 2.6 references another decision model which is shown in Figure 2.21. This model also contains one decision which reads the type of the card and the discount determined by the previous decision. Based on this information, it determines the possibility for some kind of offer. Note that the table contains overlapping rules, namely rules two and three. They both match for the input (50, 100%). Therefore, this table has a *first hit* policy, such that in this situation the output of rule two will be returned.

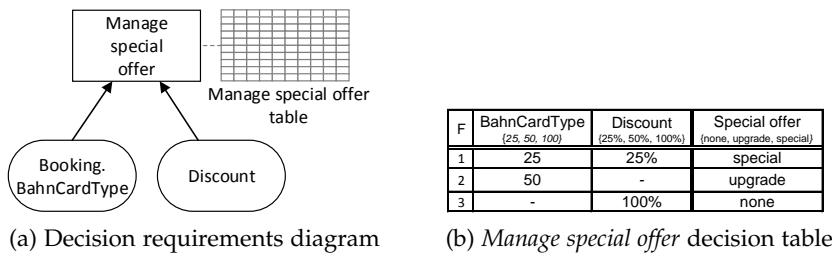


Figure 2.21: Decision model referenced by the task *Manage special offer* in Figure 2.6

It should be noted that a decision task does not need to be followed by a split gateway as required by decision fragments. For example, it may also be integrated into the process model as illustrated in Figure 2.22. In this case, the outputs of the decision are processed im-

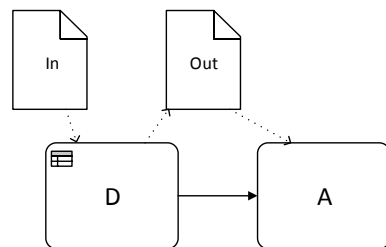


Figure 2.22: Decision task directly followed by another task reading the output of the decision

PLICITLY by the succeeding task such that it is not possible to conclude from the model how the process reacts to different results of the de-

cision. Hence, a decision-aware business process model is defined as follows:

Definition 21 (Decision-aware process model). A process model $m = (N, DO, V, C, CF, DF, \alpha, \xi, \delta)$ with set $A \subseteq N$ of activities and set $A_D \subseteq A$ of decision tasks is decision-aware if and only if $A_D \neq \emptyset$, i. e., m contains at least one decision task. \diamond

2.4 RELATED WORK

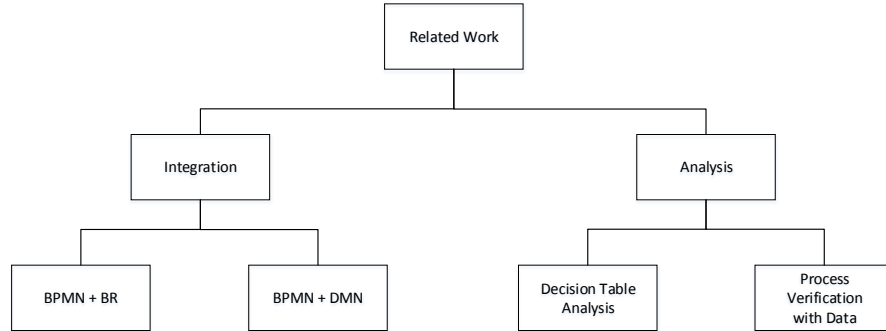


Figure 2.23: Classification of related work

This section discusses literature that is related to the work described in this thesis. The overall classification of related work is shown in Figure 2.23. Since the work of this thesis is about the analysis and verification of the integration of process and decision models, the related work is divided into two categories. On the one hand, approaches for the integration of models or standards for the expression of decision logic and BPMN are described, where decision logic is either expressed through business rules (BRs) or DMN models. On the other hand, analysis and verification methods are investigated, which focus either mainly on decision model analysis, or on processes with explicit consideration of data or decisions.

2.4.1 Integration of Processes and Rules

It is a long and well known fact that business processes involve decision-making. Such decisions have traditionally been made based on *business rules*, which is a “statement of the action to be taken when a specific set of conditions are true” [58, p. 7], or a “logical statement that allows a conclusion to be drawn from a set of conditions” [52, p. 43]. In that sense a business rule can be understood as one row/rule of a decision table in DMN. Indeed, DMN decision models can be understood as models that organize, structure and illustrate business rules [58]. Nonetheless, the proper definition and maintenance (e.g., involving ontologies) of business rules can become quite complex,

such that the OMG developed a corresponding standard, the Semantics of Business Vocabulary and Rules (SBVR) specification [93]. SBVR considers a business rule as a “rule that is practicable and that is under business jurisdiction” [93, p. 98], where being under business jurisdiction means that the organization that the rule belongs to has the freedom to change it. Certainly, there can also be rules that are imposed by the organization’s environment. However, for those rules the organization would then have to formulate corresponding internal rules, in order for them to count as *business rules*.

On the one hand, separating processes and decisions, for example via BPMN and DMN (or SBVR) satisfies the principle of the separation of concerns, recommending to divide a problem into small modules and to apply specialized problem solving methods for each module [2]. On the other hand, this requires the correct integration of the two types of concepts.

A literature survey by Imgrund et al. [89] found that there are 78 relevant research contributions between 2004 and 2016 that deal with the integration of business rules and business process management. Moreover, the lion’s share of papers focusses on the development of methods for the design and modeling of business rules as part of BPM. This is mostly due to the fact that in order to “integrate” business rules, organizations used to supplement their BPMN process models with simple textual annotations of such rules [26], such that the importance of the combination and alignment of business rules and BPM is emphasized [28, 50]. In this regard, zur Muehlen et al. [41] investigated the most suitable combination of a set of process modeling and a set of rule modeling languages based on representation theory, measuring how well they can represent a real-world domain. They come to the conclusion that the combination of BPMN and Simple Rule Markup Language (SRML) [18] is most suitable.

In [41] it is also noted, however, that rule and process modeling languages have a conceptual overlap, such that it might not always be clear in which situation to apply which language. With respect to this, [116] proposes a methodology for the integrated modeling of business processes and business rules, while [48] suggests best practices for a proper integration. Both contributions take a rather high-level perspective, however, focussing on steps such as process and rule identification and elicitation. [91] even proposes an automated approach for the generation of business process models with business rules from entity-relationship diagrams [4], and [71] presents an algorithm which specifies how to translate the SBVR vocabulary, structural and operational rules into BPMN and DMN elements.

All of those approaches do not focus on concrete consistency issues that might arise due to the integration. [47] describes a preliminary framework for the consolidation of existing business process models and business rule descriptions as well as mapping methods based on

BPMN and SBVR that assist in identifying inconsistencies between the two. However, the approach is rather preliminary and high-level and no concrete inconsistency issues are mentioned.

The only work that describes a formal integration of business processes with business rules was presented by Kluza et al. in [105]. The approach aims at ensuring data type consistency, meaning that the data types employed in rules must also be available in the processes that apply these rules. The authors assume that rules can be integrated with processes via conditional flows, events, and decision tasks. For that, certain integration rules are formally specified. Especially the integration rules formulated for decision tasks succeeded by a split gateway are related to this thesis. Given that the decision task calls a rules engine for execution, on the one hand, the authors require that all input attributes needed by the rules engine are available in the process model. On the other hand, all outgoing sequence flows of the succeeding split gateway should use the same attributes that are output attributes of the decision task. These two requirements are very similar to the definition of structural consistency between decision fragments and decision models that will be discussed in Section 6.2. However, [105] does not specify any behavioral rules that would ensure that the process model can actually handle the various outputs of the rules engine.

2.4.2 *Integration of Processes and Decisions*

Suchenia et al. [115] identified 13 papers that deal with the “interoperability” of BPMN and DMN models. Most of those papers are concerned with the exemplary integration of the two kinds of models and its benefits. Fewer deal with concrete inconsistency issues that may arise due to the integration. Both will be discussed in the following.

In [68] a proof of concept design of a combination of BPMN, DMN and Case Management Model and Notation (CMMN) models is described, while [65] and [83] present a framework that supports the integrated modeling of healthcare processes and decisions. This framework deals with high-level steps such as design, implementation and enactment. [86] proposes a Decision as a Service design for decision-aware information systems, that is supposed to improve the separation of concerns of processes and decisions. With a service-oriented design the processes only know about the interface and are agnostic of the concrete structure and implementation of the decisions.

De Smedt et al. [84, 85] argue for the use of holistic decision models in association with process models. Holistic decision models are supposed to be reused throughout the process by different decision tasks. Therefore, one decision model includes all the decisions made within a process. This differs from the use of decision fragments as

described in Section 2.3, which allows independent decision models to be referenced by the different decision tasks of the same process.

[104] discusses three different process and decision modeling approaches. A *process-centric approach* understands decisions as local concerns that are taken at exclusive split gateways. In that way, they correspond to the decision fragments of decision-aware process models described in Section 2.3. This approach requires the process to ensure that the decision is supplied with all the necessary inputs, and that all of the decision's outputs are dealt with by the outgoing branches of the split gateway. This corresponds to the decision deadlock freedom criterion that will be discussed in Section 6.3. However, it is stated that "the approach convolutes decision logic with routing logic and the distinction between the two is not always clear" [104, p. 248]. This statement is not fully comprehensible since through the use of decision models the decision logic is *not* part of the process model anymore, only the routing logic of the split gateway is used to process the decision outputs. Also, the authors state that this is the most prevalent approach in literature, with publications such as [66, 68, 75, 77, 80, 94].

The second approach for process and decision modeling discussed by [104] is the *process-extraction approach*. Here, decision logic and structure are identified in the process model and logs so as to externalize them and redesign the process model accordingly. This may lead to a convolution of process and decision logic. Examples are given in [60, 62, 63, 110].

The last approach mentioned in [104] takes a different perspective compared to the other two. It is called *decision-centric approach* that designs *holistic* decision models, which were already described above in the context of discussing the work by De Smedt et al. [84, 85]. Here, the "decisions are put above the process" [104, p. 249], and the process model is designed around the decision model. The authors emphasize that any process can call the decision model and all of its sub-decisions, instead of just coupling a single process and decision model. Furthermore, this approach will make the process model automatically consistent with the provided decision model. This will be further elaborated in the following.

In [69, 88, 102, 103] inconsistency issues are identified that can arise in situations where a decision model is called from a single point in the process or when a decision model is called multiple times in the process to obtain results from its various sub-decisions. The authors define seven inconsistencies that will be summarized and discussed in the following.

The first is *exclusion of decision outcomes*, which arises if not all outcomes of the decision are represented by the control flow of the process, given that the process redirects its control flow based on the decision outputs. Therefore, this incompatibility corresponds to the

output coverage condition of the decision deadlock freedom criterion that was already described in [77, 80], and which will be presented in Section 6.3 of this thesis.

Next, the inconsistency *inclusion of decision logic in the process* describes situations in which decision logic is modeled in the process model as opposed to the decision model. While this is not an inconsistency per se anyway, it is in any case excluded by the understanding of decision-aware process models in this thesis.

The *exclusion of intermediate results* inconsistency occurs for processes that rely on intermediary results of the decision model, i.e., results from its sub-decisions, without explicitly invoking those sub-decisions. Since in this thesis it is assumed that the process model only relies on results from the decision model's top level decision, this inconsistency cannot occur by definition in the context of this thesis.

An inconsistency due to the *inclusion of process-unrelated sub-decisions* arises in the opposite situation: a sub-decision is invoked from the process although its result is never needed. Again, since in the context of this thesis only top-level decisions can be invoked this issue cannot occur.

The *unsound ordering of decision hierarchy* inconsistency occurs when the order of the (sub-)decisions is not matched by the order of calling them in the process. For example, a sub-decision may be called after the top-level decision by the process model, so that the process model does not respect the decision hierarchy of the decision model. Again, since in the context of this thesis only top-level decisions can be invoked this issue cannot occur.

The *exclusion of sub-decisions affecting control flow* inconsistency is very similar to the *exclusion of intermediate results*. The only difference is that here the process' control flow (not a data object) relies on the sub-decision's results without invoking it. Again, since in the context of this thesis only top-level decisions can be invoked this issue cannot occur.

Finally, an *absence of input data* inconsistency arises when the process model does not supply all the required input data to the decision model when calling it. This inconsistency is prevented by *structural consistency* described in [80] and also in Section 6.2 of this thesis.

A comparison of the (in)consistency criteria described in [103] and in this thesis (based on [77, 80]) is shown in Table 2.1. This table lists the consistency criteria of both approaches and demonstrates equivalences if possible. Note that only the basic consistency criteria described in Chapter 6 of this thesis are listed there, but not the ones described in Chapter 7 and Chapter 8, since those can be considered as variations or specializations of the basic criteria. The last two rows of the table mention consistency criteria that are established in [77,

80] (and also in this thesis), but for which there are no corresponding equivalent criteria in [103].

Table 2.1: Comparison of the (in)consistency criteria described in [103] and in this thesis (based on [77, 80])

Hasić et al. [103]	Batoulis et al. [77, 80]
Exclusion of decision outcomes	Decision deadlock freedom (output coverage)
Inclusion of decision logic in the process	Excluded by the requirement of decision-aware process models to outsource their decision logic to decision models
Exclusion of intermediate results	Excluded by definition
Inclusion of process-unrelated sub-decisions	Excluded by definition
Unsound ordering of decision hierarchy	Excluded by definition
Exclusion of sub-decisions affecting control flow	Excluded by definition
Absence of input data	Structural consistency
-	Decision deadlock freedom (table completeness)
-	Dead branch absence

Finally, Posenato et al. [109] describe scenarios in the healthcare domain where process execution is subject to temporal constraints. If a currently running process instance is at risk of violating such constraints given that it selects some long running activities or branches for execution, those branches must be forbidden, by forbidding the respective outcomes of the (manual) decision that would lead to the execution of those branches.

2.4.3 DMN Decision Model Analysis

The first formal considerations regarding DMN decision tables were published by Calvanese et al. in [64, 97]. The authors present formal semantics of DMN decision tables including rule semantics and hit policies. Also, algorithms for detecting missing and overlapping

rules, and an algorithm for merging rules to simplify the table were described. The basic idea of those algorithms of using a geometric interpretation of decision tables has been reused by Batoulis and Weske [96] and also in this thesis for determining the input/output behavior of DMN decision tables, described in Chapter 4.

Missing rules can be detected by looking at the data types of the input variables and then checking if there are values in the domain of those data types that are not covered by any decision table rule. However, sometimes some of those values may never occur in reality. For example, the data type of an input variable representing the weight of an object would probably be integer or double. And for most application domains the possible weights have a certain limit. Therefore, in [82], Calvanese et al. propose a formalization of decision requirements diagrams and of their integration with background knowledge in the form of ontologies. In that way, the table completeness check can be restricted to values that are possible according to the ontology.

In [98, 99, 113] analysis efforts to check for inconsistencies in DMN decision table rules are described. The approach is based on building a rule base from decision table rules and domain knowledge. Then, algorithms for finding inconsistencies such as contradictions between the rules of the decision table and the rules derived from the domain knowledge can be applied.

Hasić et al. [87, 111, 112] define complexity metrics for decision requirements diagrams (DRDs) and decision tables. Examples of metrics for DRDs include the number of decisions and the number of elements in total. For decision tables, metrics such as the number of possible input value combinations, the number of possible output values and the hit policy are mentioned, where tables with overlapping rules are considered more complex, and multi-hit policies are more complex than single-hit policies. Furthermore, Bossuyt and Gailly [81] measure a complexity decrease of the process model when outsourcing decision logic to a decision model.

Finally, also optimizations of decision-making during process runtime are discussed. In [70] the authors argue that sometimes a decision can be made with only a partial set of inputs available, for example, when the outcome would not change given the complete set of inputs. This can save costs if acquiring the missing inputs is expensive.

2.4.4 *Process Verification with Data*

Calvanese et al. give an overview of research on data-aware business processes in [55] and Montali et al. [74] emphasize that most analysis methods for business process models only consider their control flow according to the classical notion of soundness (cf. Definition 13). Therefore, they introduce a notion of data-aware soundness that in-

cludes the interaction of the process with a relational database. However, no effective method for the verification of this notion is offered.

Also Sadiq et al. [22] argue that including the data of the process in the analysis and verification is critical. They classify the data used in processes into different types such as operational data used by activities and decision data used to make choices at split points. Moreover, they describe data flow validation issues such as missing and redundant or unused data.

Sidorova et al. describe so called conceptual workflow nets with data (WFD nets) [43, 49]. A WFD net is a workflow net extended with transition guards in the form of logical predicates, and read/write/delete operations for the transitions. Therefore, WFD nets differ from workflow nets through the extension of transitions with the following four aspects. When firing, transitions can: read from a finite set of data elements; write a finite set of data elements; and delete a finite set of data elements; lastly, a transition can only fire if its guard—being a predicate depending on the data elements read by the transition—evaluates to true.

Moreover, the WFD net is called conceptual because it does not operate on concrete values for the data elements. Rather, data elements can be *defined* or *undefined*, and they are undefined by default. If a transition writes a data element, it will be defined, and if it deletes it, it will again be undefined. The guard of a transition is undefined as long as at least one of the data elements it depends on is undefined. When all of its data elements are defined, the guard will evaluate to *both* true and false.

In that way, the authors are able to check the *may-* and *must-soundness* of a workflow net with data without considering concrete data values. A net is *may-sound* if it is sound for certain evaluations of the guards (either true or false) and *unsound* for others. If the net is sound for all possible evaluations of the guards, it is *must-sound*. If no evaluation of the guards can make the net sound, it is *unsound*. Hence, the authors are able to determine if a concretization of the conceptual WFD net with concrete data values *can* lead to a sound workflow net or not.

Therefore, the approach may give an initial idea about the soundness of the workflow or process that is investigated, but cannot yield definite results. Also, it cannot express how different data variables depend on each other, which is important for analyzing processes with decision logic. For example, it cannot express that a discount is only 25% if the BahnCard is of type 25. Moreover, they assume data elements with finite domains.

Allowing the data value of the process to come from infinite domains will lead to problems regarding the soundness verification of that process. This is because the process can take on infinitely many states, one for each different assignment of values to its data vari-

ables. Knuplesch et al. [37] propose an approach to deal with “large domains”, which can actually also be applied to infinite domains.

The authors aim to verify data-aware compliance rules that the process must adhere to, such as “After confirming an order of a non-premium customer with piece number of at least 125,000, premium status should be offered to the customer” [37, p. 333]. Here, the variable representing the piece number is of type integer and is therefore infinite, and checking if the process complies to the rule for every possible piece number is impossible.

For this reason, the authors do not check the rule for every possible value but only for representative values. A value is representative of other values, if the process behaves in the exact same way for all of those values. The representative values are inferred from gateway conditions and compliance rule conditions. For example, if a split gateway selects two different branches if the piece number is greater or less than a certain value, and then behaves the same for those two sets of values, it is sufficient to check the process for one value of each set. While this is a useful step towards incorporating infinite data domains in the analysis, the authors do not mention how to support data update operations, and also decision logic is not considered.

[106, 107] employ the same idea of using representative values, in this case to check the soundness of decision-aware process models. The approach starts from the process being modeled as a data Petri net (DPN) [108], which is similar to a WFD net described above. Note, however, that in [106, 107] concrete data values are considered. To check the soundness of decision-aware process models, the authors translate DMN decision tables referenced by decision tasks in the process models to transitions in the DPN. For each rule in the table a transition in the DPN is created that implements that rule, i. e., the transition’s guard has the same conditions as the rule, and writes the rule’s output into a corresponding variable.

To check the soundness of the DPN it is translated into a colored Petri net, which allows the authors to conduct a reachability graph analysis using representative values that are derived similar to [37]. They also describe how they can verify the various notions of soundness defined in [77] and also in Chapter 8 of this thesis. However, they restrict themselves to DMN decision tables with a *unique* hit policy, and do not offer an efficient method to deal with tables containing overlapping rules.

Part II

FORMAL FRAMEWORK FOR PROCESSES AND
DECISIONS

ON THE SEPARATION OF CONCERNS OF PROCESSES AND DECISIONS

This chapter argues in Section 3.1 that BPMN is not suited for the specification of decision logic, violating the principle of the separation of concerns. This claim is backed by an analysis of nearly 1000 real world process models, revealing that decision logic is often hard-coded in process models via three frequently occurring control-flow-based decision patterns, defined in Section 3.2. Finally, in Section 3.3 statistics about the usage of split gateways and the frequency of the decision patterns are reported. The work presented in this chapter has partly been published in [60].

3.1 PROCESS AND DECISION MODELING IN THE REAL WORLD

Nowadays, business process management and modeling is a widely used means for companies to document, control, automate, measure, and improve their processes effectively. During process execution, often decisions have to be made that impact the correctness, performance and compliance of these processes [57]. Based on an analysis of industry process models it was shown that these decisions handle choices such as the assignment of resources to activities, i. e., which knowledge worker should work on a particular activity, or determine which one of several alternative branches should be selected in order to continue the process. Especially, in the insurance and banking domains, regulatory compliance highly influences process execution by specifying which guidelines must be followed.

The analysis of 956 real world process models from the insurance, banking and health care domains revealed that the logic that decisions are composed of is frequently encoded in process models. This leads to models that are hard to read, implement and maintain. While BPMN has the means to describe how the outcome of a decision should be handled, it is not meant to represent the details of the decision logic leading to such outcome. This becomes obvious when looking at the abstract example in Figure 3.1, where decision logic is encoded through a series of exclusive split gateways, resulting in a spaghetti-like model.

Thus, decision logic modeling is out of scope for BPMN. Rather, dedicated decision modeling concepts such as decision tables [35] and other methods [39, 54] should be employed to represent complex decisions in an appropriate manner. For instance, the Decision Model and Notation (DMN) standard [61] resulted from an effort undertaken

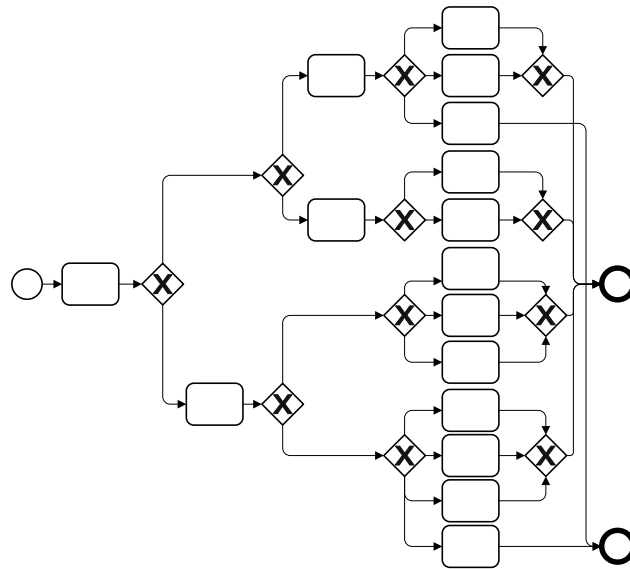


Figure 3.1: Misuse of BPMN for decision logic modeling

by the OMG to provide a standardized way of expressing decisions. DMN is designed to complement BPMN to achieve a “separation of concerns” [2] between process and decision logic modeling based on the actual scope of both modeling techniques.

On the one hand, BPMN provides extensive means to represent process logic. This includes primarily the activities that need to be performed during the process, their order and the data that is consumed and produced by them. On the other hand, DMN represents the decisions made during the process. Obviously, this includes the logic of these decisions, but also the data that is required to make the decision and documentation on what knowledge or guidelines the decision is based on. Due to a lack of proper integration between these two worlds, organizations misused BPMN constructs to express decision logic in process models.

A separation, however, provides multiple advantages. Already in 1972, Parnas argued about the benefits of a “separation of concerns” in system and software design [2]. This separation is mainly achieved by dividing a problem into smaller modules and to use for each module a specialized problem solving concept. Thereby, easier maintenance, less complex systems, reusability, flexibility, shortened development time, comprehensibility, and reduced inter-dependencies are achieved. With regard to process and decision modeling, a separation of the two worlds can provide the same benefits. Outsourcing the decision logic to a dedicated decision model reduces the complexity of the process model and increases the precision, readability, and maintainability of both process and decision models. Moreover, process and decision logic can be changed independently from each other leading to lower changing times and costs. This is especially important because process models are considered stable in general and

only need to be adapted if the business changes. Decision models, on the other hand, are required to be dynamic to react fast and flexibly to temporary situations.

For instance, consider the time after natural disasters like hurricanes in the United States. Within two to three weeks after the disaster, insurance companies have the highest increase in new customers. Usually, the companies check each applicant whether they are eligible to be accepted as a new customer based on a set of decisions. Since the time-frame after a disaster is very short and not each applicant can be checked properly without loss of applicants, insurance companies often change their policies such that a certain number of applicants are accepted without a detailed check. Therefore, in these situations, a fast reaction is required and changing a process model comprising the decision logic is not an option since it is too time-consuming. Having the decision logic separated from the business logic allows the change of rules indicating when to accept a new customer and therefore accept most or even all of the applicants. To do so, for instance in decision tables, only the values of some fields need to be adapted. Compared to complete process model structure changes, the difference in complexity and waiting time is significant. After the end of the customer rush, the decisions are changed back to normality. Finally, separating the decision logic from process model logic allows reusability of the decision model in multiple decisions occurring in the same as well as different process models.

3.2 CONTROL-FLOW-BASED DECISION PATTERNS

This section describes the results of the analysis of the 956 process models mentioned in the previous section in detail. The models were extracted from process model collections of various domains: health care, banking, insurance, energy and information technology. Each collection contains between 14 and 334 models, leading to a total number of 956 process models. These models were analyzed for structures such as the one in Figure 3.1. Therefore, the focus of the investigation was on fragments of process models that contain *control-flow-based decisions*, i. e., decisions that are encoded through a combination of activities and gateways with conditions attached to the gateways' outgoing edges. 63% of the analyzed models contain such control-flow-based decisions. Some process models even contain multiple occurrences.

The analysis revealed three types of frequently used patterns used to encode decision logic based on BPMN modeling elements. These identified patterns are described and formally defined in the following. Each pattern is represented as a process fragment (cf. Definition 12). Thereby, we utilize an example fragment of a process model from the insurance domain dealing with assigning the correct discount for a customer. In the examples, for clarity reasons, we some-

times visualize only two outgoing edges of a split gateway. Yet, in practice, there can be any number of edges as covered by the corresponding formalisms. These patterns have already been formally defined in [60]. However, in the following we provide more complete and accurate definitions.

3.2.1 P_1 —Single Split Gateway

A process model fragment matching pattern P_1 contains a decision structure of an activity followed by a single exclusive or inclusive split gateway with at least two outgoing edges. Each outgoing edge of the gateway is connected to an activity. Moreover, pattern P_1 includes “optionality decisions” as a special case. These are structures in which one of the outgoing edges of the gateway is not followed by an activity. Rather, it is, for instance, directly followed by a join gateway that joins all the exclusive paths of the split gateway. Therefore, executing an activity after the split gateway is optional. In order to also consider these cases as instances of pattern P_1 , such “empty” edges are extended by τ -transitions (or activities), i. e., empty activities that do not have an influence on process execution and data.

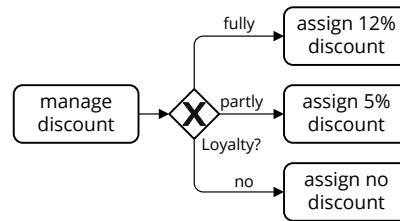


Figure 3.2: Process fragment representing a split gateway with more than 2 outgoing edges

Figure 3.2 presents a corresponding process fragment with three alternative paths at the split gateway. Depending on the modeling style, the bottom activity *assign no discount* might not have been modeled such that it would have been added as a τ -transition instead. Since the gateway is of type XOR, only one alternative can be chosen. Based on the result of activity *manage discount*, i. e., the taken decision about the customer’s loyalty, the discount assigned to the customer is set to 12%, 5%, or 0% respectively. Possible results of the decision are fixed by the annotations on the edges originating from the split gateway.

Formally, we specify pattern P_1 as follows.

Definition 22 (Pattern P_1). Let $pf = (N', DO', V', C', CF', DF', \gamma, \beta, \sigma)$ be a process fragment of process model m . Then, pf represents P_1 if

- $N' = A' \cup G' \cup E'$, where $E' = \emptyset$ (pf only contains activities and gateways),

- $|G'| = 1 \wedge G' = \{g\} \wedge |\bullet g| = 1 \wedge |g \bullet| \geq 2$ (the fragment contains exactly one split gateway),
- $|A'| = |g \bullet| + 1$ (the number of activities of pf equals the number of outgoing edges¹ of the split gateway g plus 1),
- $\exists a_s \in A' : |\bullet a_s| = 0 \wedge \forall a \in A' \setminus \{a_s\} : |a \bullet| = 1$ (a_s is the start node of pf),
- $|a_s \bullet| = 1 \wedge \forall a \in A' \setminus \{a_s\} : |a \bullet| = 0$ (activities other than the start node a_s are end nodes of pf),
- $\forall a \in A' \setminus \{a_s\}, \exists c \in C' : (g, a) \in CF' \implies \beta(g, a) = c$ (all outgoing edges of the split gateway are annotated with a condition).

◇

3.2.2 P2—Sequence of Split Gateways (Decision Tree)

A process model fragment that corresponds to pattern P2 begins with an activity followed by a split gateway that has at least two outgoing edges. Each of those outgoing edges is connected to an activity or another split gateway, with the requirement that at least one of the outgoing edges must be connected to another split gateway. In case the edge is connected to an activity, this activity is an end node of the fragment. However, if the edge is connected to another split gateway, it can again be followed by a split gateway or an activity. This proceeds iteratively until all paths reach an activity, i. e., on each path from the first split gateway to some end node of the fragment, there exists exactly one activity. This gives rise to a decision tree structure.

Figure 3.3 presents a corresponding process fragment with a total of four alternative paths after the first split gateway. Since all gateways are of type XOR, only one alternative can be chosen.

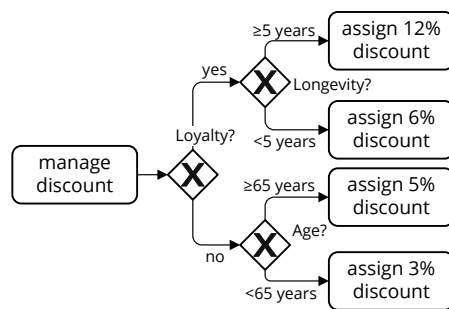


Figure 3.3: Process fragment representing a sequence of split gateways that represents a decision tree

The actual routing based on the taken decisions is distributed over two split gateways. Based on the result for the customer loyalty, the

¹ the number of outgoing (incoming) edges directly translates to the number of direct successors (predecessors) and vice versa

second routing decision is either taken based on the longevity of the customer relationship (loyal customer) or the age of the customer (non-loyal customer). Due to the dependency of a routing decision on the ones taken before, this pattern represents a decision tree. Analogous to pattern P_1 , the possible results of the decision are fixed by the annotations on the edges originating from some split gateway.

Formally, we specify pattern P_2 as follows.

Definition 23 (Pattern P_2). Let $pf = (N', DO', V', C', CF', DF', \gamma, \beta, \sigma)$ be a process fragment of process model m . Then, pf represents P_2 if

- $N' = A' \cup G' \cup E'$, where $E' = \emptyset$ (pf only contains activities and gateways),
- $|G'| \geq 2 \wedge \forall g \in G' : |\bullet g| = 1 \wedge |g \bullet| \geq 2$ (all gateways of the fragment are split gateways),
- $|A'| = (\sum_{g \in G'} |g \bullet|) - (|G'| - 1) + 1$ (the number of activities of pf equals the number of outgoing edges of the split gateways not connected to other split gateways, plus 1),
- $\exists a_s \in A' : |\bullet a_s| = 0 \wedge \forall a \in A \setminus \{a_s\} : |\bullet a| = 1$ (a_s is the start node of pf),
- $|a_s \bullet| = 1 \wedge \forall a \in A' \setminus \{a_s\} : |a \bullet| = 0$ (activities other than the start node a_s are end nodes of pf),
- $\forall g \in G', \forall n \in g \bullet : n \in A' \cup G' \setminus \{g\}$ (all successors of a gateway are an activity or a gateway), and
- $\forall a \in A' \setminus \{a_s\}, \forall g, g' \in G', \exists c \in C' : ((g, g') \in CF' \implies \beta(g, g') = c) \wedge ((g, a) \in CF' \implies \beta(g, a) = c)$ (all outgoing edges of the split gateways are annotated with a condition).

◇

3.2.3 P_3 —Sequence of Split Gateways Separated by an Activity

A process model fragment that matches pattern P_3 contains a decision structure of an activity succeeded by a split gateway with at least two outgoing edges. Each of those outgoing edges is connected to an activity or another split gateway with at least two outgoing edges. In both cases, the successor node may be another split gateway with at least two outgoing edges. Therefore, there can be activities in between gateways. Iteratively, this proceeds until all paths reach an activity that is not succeeded by a split gateway. This means, this pattern can be composed of a combination of decision trees as well as single split gateways.

Figure 3.4 presents a corresponding process fragment with a total of three alternative paths after the first split gateway. Since all

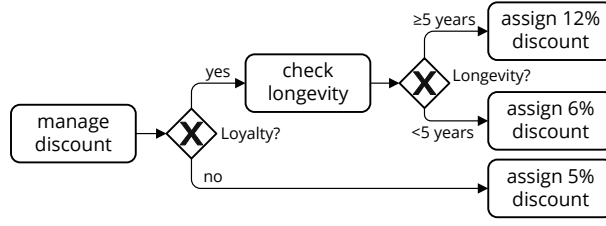


Figure 3.4: Process fragment representing a sequence of split gateways separated by an activity

gateways are of type XOR only one alternative can be chosen. In the example, the choice between 12% and 6% discount is taken based on two decisions (loyalty and longevity) while granting 5% discount is clear after the first decision for non-loyal customers.

Formally, we specify pattern P_3 as follows.

Definition 24 (Pattern P_3). Let $pf = (N', DO', V', C', CF', DF', \gamma, \beta, \sigma)$ be a process fragment of process model m . Then, pf represents P_3 if

- $N' = A' \cup G' \cup E'$, where $E' = \emptyset$ (pf only contains activities and gateways),
- $|G'| \geq 2 \wedge \forall g \in G' : |\bullet g| = 1 \wedge |g \bullet| \geq 2$ (all gateways of the fragment are split gateways),
- $|A'| \in [(\sum_{g \in G'} |g \bullet|) - (|G'| - 1) + 1 + 1, (\sum_{g \in G'} |g \bullet|) - (|G'| - 1) + (|G'| - 1) + 1]$ (the number of activities of pf equals the number of outgoing edges of the split gateways not connected to other split gateways, plus at least 1, and at most plus the number of gateways connecting to other split gateways, plus 1),
- $\exists a_s \in A' : |\bullet a_s| = 0 \wedge \forall a \in A' \setminus \{a_s\} : |\bullet a| = 1$ (a_s is the start node of pf),
- $\exists g \in G' : \bullet g = \{a_s\}$ (activity a_s is followed by a gateway),
- $\forall n \in N' : |n \bullet| = 0 \implies n \in A'$ (all end nodes are activities),
- $\forall g \in G', \forall n \in g \bullet : n \in A' \cup G' \setminus \{g\}$ (all successors of a gateway are an activity or a gateway), and
- $\forall a \in A' \setminus \{a_s\}, \forall g, g' \in G', \exists c \in C' : ((g, g') \in CF' \implies \beta(g, g') = c) \wedge ((g, a) \in CF' \implies \beta(g, a) = c)$ (all outgoing edges of the split gateways are annotated with a condition).

◇

3.3 STATISTICS ABOUT THE DECISION PATTERNS

As mentioned above, we analyzed 956 process models from various real world process model collections. This section first presents some

Table 3.1: Statistics on the usage of split gateways in the analyzed model collections

	XOR gateway	IOR gateway	Binary split	Multi split
Freq	80.83%	8.24%	79.3%	19.27%
Avg	3.7	0.15	2.74	0.23

general statistics on the usage of split gateways in these models and then shows the results of a frequency analysis of patterns P_1 – P_3 .

Table 3.1 demonstrates that the exclusive gateway is used very frequently, in more than 80% of all models, with an average of 3.7 usages per model. This is a ten times higher frequency than that of the inclusive gateway. Moreover, most split gateways are binary, i. e., they have only two outgoing edges, while one out of five gateways has more than two outgoing edges. Altogether, this demonstrates that in most process models several decisions are taken, and that most decisions have binary outcomes.

In addition to the split gateway statistics, the process model collections were also analyzed regarding the frequency of the patterns described in the previous section. The analysis was conducted automatically and therefore required syntactically correct process models. This is the case for 566 of the 956 models. The results are presented in

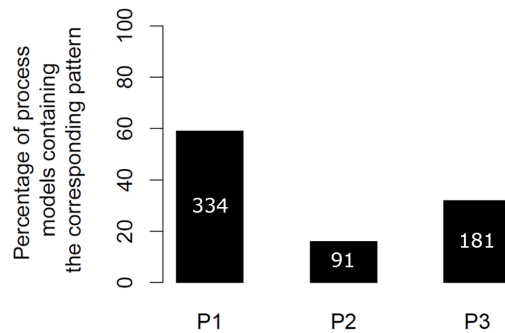
Figure 3.5: Frequencies of patterns P_1 – P_3 in real world process models

Figure 3.5 and show that pattern P_1 occurs in 59%, pattern P_2 in 16%, and pattern P_3 in 32% of all process models. In total, we observed 680 occurrences of P_1 fragments, 113 occurrences of P_2 fragments, and 362 occurrences of P_3 fragments in our set of 566 models.

These numbers show that the identified patterns are frequently used in practice. Therefore, BPMN is often misused to represent decisions and their logic. As discussed in the beginning of this chapter, a separation of concerns provides many benefits and the specification of decisions using the DMN standard complementary to BPMN can achieve such a separation. However, the integration and interplay of the two kinds of models requires extended correctness checks to en-

sure an overall sound execution of the business process. This is the topic of Part [iii](#) of this thesis.

INPUT-OUTPUT BEHAVIOR OF DMN DECISION TABLES

This chapter is concerned with the behavior of DMN decision tables, discussing how it maps a set of input values to a set of output values. In Section 4.1, it is argued that the behavior of decision tables with overlapping rules is hard to understand, making them unsuitable for certain analysis tasks. Therefore, Section 4.2 presents an algorithm to transform a table with overlapping rules into an equivalent table with exclusive rules. Afterwards, we show how decision tables can be interpreted as functions in Section 4.3, and give a least upper bound for the number of possible outputs of a table with one input column in Section 4.4. The work presented in Section 4.1 and Section 4.2 has partly been published in [96].

4.1 PROBLEM STATEMENT

Decision tables are standardized in DMN because they proved to be a suitable representation for decision logic [76]. Their understandability is based on the tabular representation of rules that map inputs to outputs. In simple cases, every given input matches not more than one rule such that this rule's output can be returned. Yet, it is also possible to design more intricate tables that contain overlapping rules. In this case, more than one rule can apply for a given input. For such circumstances, DMN offers hit policies as discussed in Section 2.2.2, which provide various conflict resolution options. For example, the *rule order* hit policy just returns the outputs of all matching rules in a list ordered by the order of the rules. However, the problem with tables containing overlapping rules is that their input-output behavior is not clearly visible anymore, thereby increasing their complexity [112]. This is because each input could match any number of rules, whose outputs are then aggregated in some way.

R	Income (k) <i>Number</i> ≥ 0	Assets (k) <i>Number</i> ≥ 0	Credit Rating {A, B, C, D, E}
1	≤ 30	≤ 30	A
2	$\in [10..60]$	$\in [10..25]$	B
3	$\in [20..95]$	$\in [40..90]$	C
4	≥ 80	-	D
5	≥ 40	≥ 85	E

Figure 4.1: DMN decision table to determine a credit rating

For example, consider the table in Figure 4.1. The table has five rules, each of which matches for certain inputs and relates them to an

output. For instance, rule 1 matches for the input (15, 10) and relates it to the output A . However, the same input is also matched by rule 2, which relates it to a different value, namely B . This means that rules 1 and 2 are overlapping, leading to ambiguities regarding the input values for which they both match: Which output value should be chosen for these input values? For this reason, the table employs a *rule order* hit policy, represented by the letter R in its upper left corner. Therefore, the table would actually relate the input (15, 10) to the output $[A, B]$.

This example illustrates that the input-output behavior of tables with overlapping rules may not be immediately clear. More precisely, it is not obvious what the possible output values of the table are, and which input values generate which of these output values. For example, the possible output values for the table in Figure 4.1 are the following:

$$A, B, C, D, E, [A, B], [C, D], [C, E], [D, E], [C, D, E],$$

which is a far from obvious by just looking at the table. Similarly, it is not obvious which output value is generated by which input value(s). For example, given the input (50, 85), one can see that it matches rule 3, but since this is a table with overlapping rules, this may not be the only matching rule. One has to keep scanning the rules until all matches are found—an error-prone process. In fact, also rule 5 matches for this input and the final output is $[C, E]$.

Altogether, decision tables containing overlapping rules have several disadvantages. On the one hand, they can be unintuitive and hard to understand for humans. On the other hand, they are also not suited for automated analysis tasks that are based on the input-output behavior of the table, such as checking their correct integration with business process models [77, 80, 88], a topic that will be discussed in Part iii. Hence, in the following section, an algorithm is described that transforms any type of DMN decision table into one that has only exclusive rules—a process called uniqueification. Since the rules of the resulting table are exclusive, the possible output values of the table are simply given by the outputs of the individual rules—no complicated combination of multiple matching rules is required—leading to a clear input-output behavior of that table.

4.2 DECISION TABLE UNIQUEIFICATION

The foundations for the uniqueification of DMN decision tables were laid in Section 2.2.4 by introducing the geometric interpretation of such tables. For example, the geometric interpretation of the table in Figure 4.1 is shown in Figure 4.2.

Given this representation, it is quite easy to see which rules match for which input. For example, for all inputs where $Income \in [10, 30]$

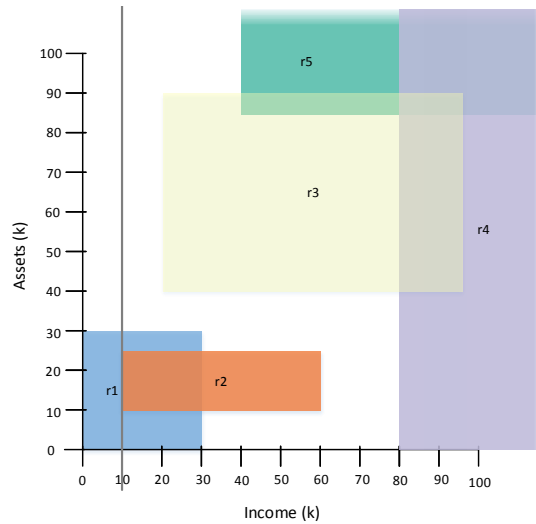


Figure 4.2: Decision table as a set of hyperrectangles corresponding to the rules of the table in Figure 4.1

and $Assets \in [10, 25]$ rules 1 and 2 match. For $Income \in (30, 60]$ and $Assets \in [10, 25]$, on the other hand, only rule 2 will match. Clearly, there can also be allowable inputs for which none of the rules match. In the geometric interpretation, these unmatched inputs are all the white “gaps” between the rectangles. This means that the table is incomplete.

Similar to the techniques of finding overlapping and missing rules in a decision table, the geometric interpretation together with the line sweeping technique (cf. Section 2.2.4) can be applied to efficiently uniqueify tables. The idea is to use the line sweeping technique to find out for which inputs which rules match. In case the table is not a *unique* hit table, it can afterwards be translated to one that is unique. This means that the rules are then exclusive, making the table not only more understandable, but also more suitable for certain analysis tasks.

Algorithm 1 shows the pseudocode of the algorithm. Initially, the algorithm is called as follows: $findMatchingRules(R, [], 0, N, [])$, where R is the set of rules of the table, $matchingIntervals$ is an initially empty list, i is an index initially set to 0, N is the number of inputs of the table, and $matchingRules$ is an initially empty list.

Given these parameters the sweep line procedure can start, which analyzes the rules of the table one dimension at a time. This requires sorting the endpoints of the intervals covered by each rule in the current dimension in ascending order (line 2). Then, the endpoints are iterated over (lines 4–12). If the current *endpoint* is a lower one (line 8), the corresponding rule is considered active, and is therefore added to the list of *activeRules* (line 9). Otherwise, if the current *endpoint* is an upper one (line 10), the corresponding rule is considered dead, and is therefore deleted from the list of *activeRules* (line 11).

Algorithm 1: findMatchingRules

Data: *rules, matchingIntervals, i, N, matchingRules*

```

1  if  $i < N$  then
2     $sortedEndpoints = rules.getSortedRulesEndpoints(i)$ 
3     $activeRules = []$ 
4    foreach  $endpoint \in sortedEndpoints$  do
5      if  $!activeRules.isEmpty()$  then
6         $matchingIntervals[i] = [lastEndpoint, endpoint]$ 
7         $findMatchingRules(activeRules, matchingIntervals, i + 1,$ 
8           $N, matchingRules)$ 
9      if  $endpoint.isLowerBound()$  then
10        $activeRules.add(endpoint.getRule())$ 
11     else
12        $activeRules.delete(endpoint.getRule())$ 
13      $lastEndpoint = endpoint$ 
14  else
15    if  $matchingRules.canBeMergedWith(\{activeRules,$ 
16       $matchingIntervals\})$  then
17       $matchingRules.mergeWith(\{activeRules, matchingIntervals\})$ 
18    else
19       $matchingRules.add(\{activeRules, matchingIntervals\})$ 
20  return  $matchingRules$  /* contains sets of rules together with
21    matching input intervals */

```

For instance, at the beginning, when $i = 0$, $sortedEndpoints = [r1(0), r2(10), r3(20), r1(30), r5(40), r2(60), r4(80), r3(95), r4(\infty), r5(\infty)]$. Therefore, in the first iteration of the for-loop $r1$ will be added to $matchingRules$. Afterwards, the current endpoint is saved as the *last-Endpoint* (line 12) and the line is swept until the next endpoint is reached (cf. Figure 4.2).

During the iteration of the for-loop, if there is at least one active rule (line 5), then the interval in which these rules are active is saved (line 6). Therefore, $matchingIntervals[i] = [0, 10]$. Note that the interval does not include the upper endpoint because for the value 10 also $r2$ would be active. Next, there is a recursive call to $findMatchingRules()$, to check the second dimension ($i + 1 = 1$), but only for the list of $activeRules = [r1]$, and only in the $matchingIntervals = [[0, 10]]$. In general, if a line is being swept across a higher dimension, it only considers rules that lie within the matching intervals of the lower dimensions. This is visualized in Figure 4.3.

In the second dimension, $sortedEndpoints = [r1(0), r1(30)]$, such that in the first iteration of the loop $r1$ becomes active (line 9). In the second iteration, the list of matching intervals is updated (line 6),

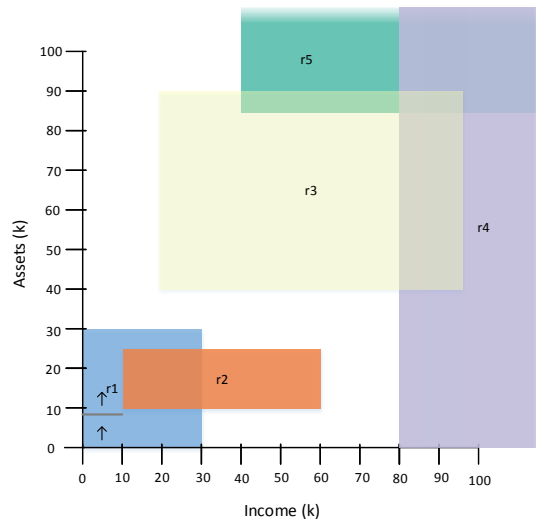


Figure 4.3: Line being swept through the second dimension in the interval $[0, 10)$

such that $matchingIntervals = [[0, 10), [0, 30]]$. Then, the procedure is recursively called again. At this point, $i = N = 2$. Hence, the else-part of the algorithm is executed (line 13), meaning that the first matching rule was found which is added to the corresponding list (line 17): $matchingRules = [\{[r1], [[0, 10), [0, 30]]\}]$.

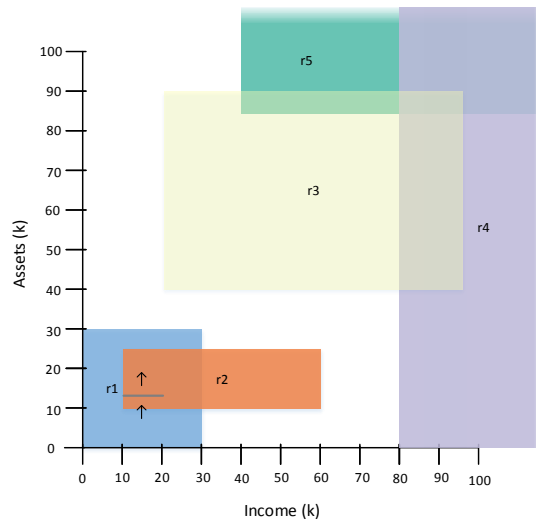


Figure 4.4: Line being swept through the second dimension in the interval $[10, 20)$

The algorithm then returns from the recursive call and deletes $r1$ from $activeRules$ because the current $endpoint$ is $r1(30)$ which is the upper bound of $r1$ in the second dimension. With that, all endpoints of the second dimension have been processed and the algorithm returns to the first dimension. This dimension was left when the current $end-$

point was $r2(10)$ (cf. Figure 4.2), such that the line sweeping continues from there.

Consequently, $r2(10)$ is added to *activeRules*, and in the next iteration of the for-loop the line in Figure 4.2 is swept until it hits the endpoint $r3(20)$. Again, a recursive call is made with *activeRules* = $[r1, r2]$ and *matchingIntervals* = $[[10, 20]]$, and the line is swept across the second dimension in that interval (cf. Figure 4.4). Accordingly, the algorithm will detect that

- $r1$ matches for the intervals $[[10, 20), [0, 10)]$;
- both $r1$ and $r2$ match for the intervals $[[10, 20), [10, 25]]$;
- $r1$ also matches for the intervals $[[10, 20), (25, 30]]$.

Coming back to the first dimension, the current *endpoint* $r3(20)$ is added to *activeRules*, and the line is swept to the upper bound $r1(30)$. Thus, in the second dimension the interval $[20, 30]$ is analyzed. After that, the current state of all *matchingRules* would be visualized as in Figure 4.5a.

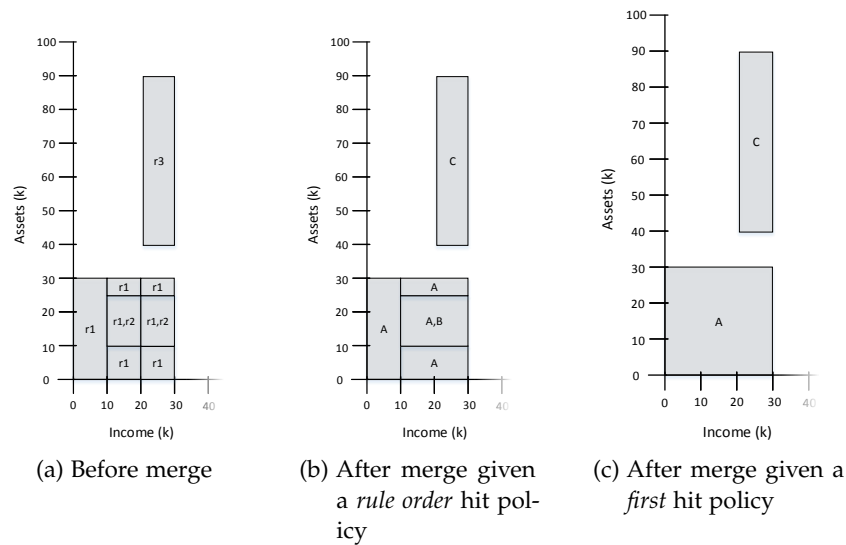


Figure 4.5: Set of *matchingRules* found so far before and after merge

Like the previous figures, it consists of rectangles, but in this case, they are labeled with the rules that match for the inputs making up each rectangle. For example, rules 1 and 2 both match for the inputs spanning the rectangle $[[10, 20), [10, 25]]$ giving rise to the corresponding label in Figure 4.5a.

This figure demonstrates that the rectangles can be optimized, since many of them cover the same interval in one dimension, and cover adjacent intervals in the other dimension. For example, the two rectangles that are labeled with $r1, r2$ are the same with respect to the

Assets dimension, i.e., they both cover the interval $[10, 25]$, while in the *Income* dimension they are adjacent, i.e., one covers the interval $[10, 20)$ and the other one covers the interval $[20, 30]$. Therefore, these pairs of rectangles can be merged into single rectangles.

As observed in [97], generally, a pair of hyperrectangles with n dimensions is eligible for merging if they overlap in $n - 1$ dimensions and are adjacent in the remaining dimension, and if the corresponding rules have the same output. However, [97] only deals with unique hit tables. This means that each hyperrectangle only represents one rule, such that the corresponding output is determined by that single rule. In case of the algorithm described here, more than one rule can belong to the same rectangle, so that the output of that rectangle depends on the hit policy of the underlying decision table.

For instance, the table in Figure 4.1 has a *rule order* hit policy. Consequently, the rectangles in Figure 4.5a labeled with $r1, r2$ will produce the output $[A, B]$. Hence, they can be merged with each other, but not with any of the other rectangles. The result of all possible merges given a *rule order* hit policy is illustrated in Figure 4.5b. To illustrate the effect that different hit policies have on these merges, assume that the table has a *first* hit policy instead. In this case, all of the rectangles in the lower left corner in Figure 4.5a will produce the same output (A). Therefore, the result after merging would be as shown in Figure 4.5c.

The merging of rectangles is reflected in the else part in Algorithm 1 (lines 13–17). As soon as a new rectangle (i.e., matching rule) is discovered by the sweep line method, a check is conducted if it can be merged with other rectangles found so far (line 14). As described in the previous paragraph, this requires the computation of the output that is produced by the newly found rectangle based on the table's hit policy. If rectangles can be merged, the result is put into the list of *matchingRules* instead of its parts (line 17). Hence, after the line has been swept across the second dimension in the interval $[20, 30]$, the list of *matchingRules* will actually look as visualized in Figure 4.5b, because three rectangle pairs can be merged.

Ultimately, after all *sortedEndpoints* have been processed, the list of *matchingRules* discovered by Algorithm 1 will contain 15 elements, ending with the element $\{[r4, r5], [(95, inf), [85, inf)]\}$, as visualized in Figure 4.6. This figure illustrates that the original set of five overlapping rectangles was now partitioned into a set of 15 exclusive (or unique) rectangles, covering the same input space as before. These unique rectangles can be translated back into a decision table. This table is shown in Figure 4.7. It has 15 exclusive rules and is therefore a unique hit table as indicated by the letter U in the upper left corner. It has the same input-output behavior as the table in Figure 4.1. However, due to the fact that all of its rules are exclusive it is more easily comprehensible and the set of possible outputs of the table is

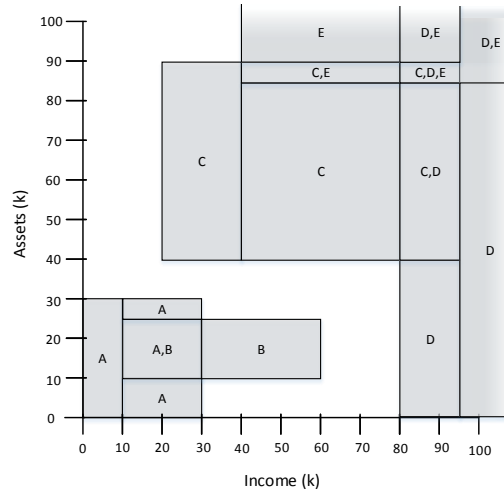


Figure 4.6: Decision table as a set of hyperrectangles

U	Income (k) <i>Number</i> ≥ 0	Assets (k) <i>Number</i> ≥ 0	Credit Rating <small>(A,B,C,D,E)/(A,B)/(C,D)/(C,E)/(D,E), <small>(C,D,E)</small></small>
1	< 10	≤ 30	A
2	$\in [10..30]$	< 10	A
3	$\in [10..30]$	$\in [10..25]$	A,B
4	$\in [10..30]$	$\in [25..30]$	A
5	$\in [20..40]$	$\in [40..90]$	C
6	$\in [30..60]$	$\in [10..25]$	B
7	$\in [40..80]$	$\in [40..85]$	C
8	$\in [40..80]$	$\in [85..90]$	C,E
9	$\in [40..80]$	> 90	E
10	$\in [80..95]$	< 40	D
11	$\in [80..95]$	$\in [40..85]$	C,D
12	$\in [80..95]$	$\in [85..90]$	C,D,E
13	$\in [80..95]$	> 90	D,E
14	> 95	< 85	D
15	> 95	≥ 85	D,E

Figure 4.7: Uniquified table derived from the table in Figure 4.1

immediately visible. Furthermore, this representation of the table is more suitable for certain analysis tasks as will be discussed in Part [iii](#).

4.3 DECISION TABLES AS FUNCTIONS

A DMN decision table, through its rules and its hit policy, defines a (partial) function. The domain of this function is the Cartesian product of the domains of the inputs of the table, and the function’s codomain is *based on* the domain of the output of the table, i.e., it can be derived from the output’s domain, as explained below. Furthermore, the function defined by the decision table may be partial. This is the case if the table is incomplete, meaning that there are allowable input values for which no rule matches, such that this input cannot be mapped to any output. For example, the table in Figure 4.7 defines a partial function since, for instance, there is no rule for the input (10, 40).

The domain of this partial function is $\mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}$ because both of the table's inputs are real numbers greater than or equal to zero. Hence, in general, the domain of a decision table function is given by the set IV , as defined in Definition 18.

The codomain of a decision table function is not so obvious, however. For single-hit decision tables the codomain could simply be given by the set of output values OV of the decision table (cf. Definition 18). In case of multi-hit tables, the situation is more complex because combinations of values in OV can be returned. Furthermore, they may be ordered in a certain way, depending on the concrete multi-hit policy of the table. Therefore, the codomain can be composed of several multisets¹ from the set OV , where each multiset has a total ordering determined by the hit policy. As a result, the function characterized by a decision table is defined as follows:

Definition 25 (Decision table function). Given a decision table dt with a set of input values IV and a set of output values OV , let OV_{MS} be the set of multisets of OV . Also, given the set of rules R of dt , let $output(r)$ be the output value of rule r , where $output(r) \in OV$, for $r \in R$. Then, let $OV' = +_{r \in R} \{output(r)\}$ be the multiset composed of the output values of all rules.² Note that $OV' \in OV_{MS}$. Consequently, the function t underlying dt is given by

$$t : IV \rightarrow 2^{OV'} \setminus \emptyset,$$

where each $ov' \in OV'$ may be totally ordered according to the hit policy p of dt if $p \in \{output\ order, rule\ order\}$. Note that this function characterizes both single- and multi-hit tables because in both cases the table's image (i. e., the elements of the codomain it actually maps to) will be some subset of the codomain given above. \diamond

R	Input1 <i>Number ≥ 0</i>	Input 2 <i>Number ≥ 0</i>	Output <i>(A, B, C)</i>
1	$\in [0..6]$	$\in [2..8]$	A
2	$\in [1..5]$	$\in [1..7]$	B
3	$\in [2..8]$	$\in [0..6]$	B

Figure 4.8: Decision table with three rules and two different output values

For example, consider the table in Figure 4.8. In this instance, $OV' = \{A^1, B^2\}$, where the superscript of each element denotes its multiplicity. The powerset (excluding the empty set) of OV' is given by $\{\{A^1\}, \{B^1\}, \{A^1, B^1\}, \{B^2\}, \{A^1, B^2\}\}$, which is this table's codomain. All the subsets of this codomain that contain more than one element are totally ordered according to the order of the rules, since the table has a *rule order* hit policy. For example, consider the input (1, 2). This

¹ Multisets are necessary because different matching rules can have the same output value.
² The + symbol represents multiset addition as defined in Definition 6.

input will match rules 1 and 2, such that the totally ordered subset $\{A, B\}$ is returned, where $A < B$, meaning that A precedes B in the ordering. But it could also be $B < A$, if the rules were reversed. Note, however, that the codomain cannot contain the same subset with different orderings. This also makes sense from the point of view of the underlying decision table, since if a set of rules overlap, the hit policy will ensure a certain ordering of the returned output.³

4.4 MAXIMUM NUMBER OF OUTPUTS OF A DECISION TABLE

The considerations in the preceding section give rise to an interesting question: given a DMN decision table with r rows and c input columns, what is the maximum possible number of distinct outputs for this table? The *upper bound* of the number of distinct outputs can be easily inferred from the decision table function in Definition 25 by computing the cardinality of its codomain.

Lemma 4.4.1. Given a DMN decision table dt whose underlying function is $t : IV \rightarrow 2^{OV'} \setminus \emptyset$, the upper bound of the number of distinct outputs of dt is the following:

$$\prod_{ov' \in OV'} (m(ov') + 1) - 1,$$

where $m(ov')$ is the multiplicity of element ov' . This formula yields the cardinality of the powerset of the multiset OV' , excluding the empty set.

Proof. According to Definition 25, $2^{OV'} \setminus \emptyset$ is the codomain of the function underlying a decision table. Since a table can only produce outputs that are in this codomain, the upper bound of the number of distinct outputs of the table is given by the cardinality of the codomain. \square

For example, given a decision table with rules $R = \{r_1, r_2, r_3\}$, such that $output(r_1) = A$ and $output(r_2) = output(r_3) = B$ (cf. Figure 4.8), the multiset OV' composed of the output values of all rules is $OV' = \{A^1, B^2\}$. The powerset (excluding the empty set) of OV' is given by $\{\{A^1\}, \{B^1\}, \{A^1, B^1\}, \{B^2\}, \{A^1, B^2\}\}$, and its cardinality equals $(1 + 1) \times (2 + 1) - 1 = 5$ as determined by the formula above.

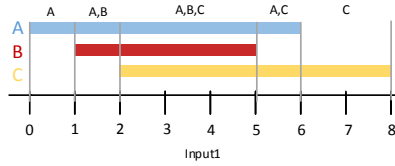
As another example, assume the rules in R all have different outputs such that $OV' = \{A^1, B^1, C^1\}$. In this case, OV' is a regular set, whose powerset (excluding the empty set) contains $2^{|OV'|} - 1 = 7$ elements. Note that this number can also be determined by the formula of Lemma 4.4.1. Therefore, this formula provides a general way of determining the maximum possible number of distinct outputs of any DMN decision table.

³ The *collect* policy “returns all hits in arbitrary order” [76]. We assume that this order will always be the same for a particular table.

While Lemma 4.4.1 provides an upper bound of the number of outputs, it is not necessarily the *least* upper bound. For example, consider the decision table in Figure 4.9a. This table has three rules, one input column and a *rule order* hit policy, and each rule has a different output. In Figure 4.9b the table's geometric interpretation is shown,

R	Input1 <i>Number ≥ 0</i>	Output {A, B, C}
1	∈ [0..6]	A
2	∈ [1..5]	B
3	∈ [2..8]	C

(a) Decision table with five possible outputs



(b) Geometric interpretation of the table on the left

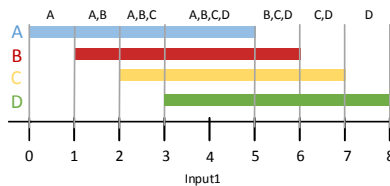
Figure 4.9: Example of a decision table with one input column, and its geometric interpretation

which consists of three overlapping one-dimensional hyperrectangles (i. e., lines). The vertical lines in this figure partition those hyperrectangles into five unique segments. They are unique in terms of the lines (or rules) that lie within that segment, and the output corresponding to each segment is displayed above. Hence, this table falls short of achieving the upper bound of the number of outputs of a table with three different output values, which would be $(1 + 1)^3 - 1 = 7$ according to Lemma 4.4.1.

As another example, consider the table in Figure 4.10. It is very similar to the previous table, except that it contains an additional rule with an additional output value. Its geometric interpretation in Figure 4.10b shows that the hyperrectangles can be partitioned into seven unique segments, where each segment has a different output. Again, this number is lower than $(1 + 1)^3 - 1 = 15$, which would be the maximum number of distinct outputs according to Lemma 4.4.1.

R	Input1 <i>Number ≥ 0</i>	Output {A, B, C, D}
1	∈ [0..5]	A
2	∈ [1..6]	B
3	∈ [2..7]	C
4	∈ [3..8]	D

(a) Decision table with seven possible outputs



(b) Geometric interpretation of the table on the left

Figure 4.10: Another example of a decision table with one input column, and its geometric interpretation

These two examples suggest that in case of only one input column, the upper bound cannot always be reached due to the fact that the input space can only be partitioned into a certain number of unique segments, namely five segments in case of three rules, and seven seg-

ments in case of four rules. This observation can be generalized in the following Lemma:

Lemma 4.4.2. Given a DMN decision table dt with one input column and r rules, let $seg(dt)$ be the number of unique segments that its hyperrectangle representation can be partitioned into. It holds that $seg(dt) \leq 2r - 1$.

Proof. The proof is carried out by induction. As the base case, assume that $r = 1$, so that the decision table has one rule. The hyperrectangle representation of any such table will look like illustrated in Figure 4.11. Since the table has only one input column, the hyperrect-

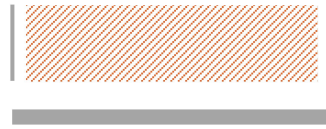


Figure 4.11: Induction base case

angle is a line that is displayed at the bottom of the figure. At the left and right edges of that line, two vertical lines show the borders of the unique segment that this line creates, and the segment is colored in a certain pattern. Obviously, there is only one segment (or pattern) that can be created with a single rule, such that $seg(dt) = 1 \leq 2r - 1 = 1$.

For the inductive step, we can assume that for a decision table with $r = n$ rules it holds that $seg(dt) \leq 2n - 1$. The segmentations of the hyperrectangle interpretation are shown in Figure 4.12. The lines



Figure 4.12: Induction hypothesis

representing the n rules of the table are omitted intentionally because their exact configuration is unknown. But what can be assumed for the inductive step is that these rules cannot be partitioned into more than $2n - 1$ unique segments. These segments are shown as different patterns in the figure.

We assume that the segments do not contain any gaps between them, i. e., the dots in Figure 4.12 must represent actual patterns and no white space. Otherwise, a dummy pattern will be added to fill the gap. This is possible because if there are gaps there will be less than $2n - 1$ segments. For example, assume that there is indeed a gap in Figure 4.12. This would mean that the n rules could be divided into p rules left of the gap and q rules right of the gap, where $n = p + q$, such that those two sets would not contain any gaps. Consequently, based on the induction hypothesis, there would be at most $2p - 1 + 2q - 1 = 2(p + q) - 2 = 2n - 2$ patterns or segments, which is less than $2n - 1$.

In such cases the gap can be filled with a dummy pattern, such that it holds that there are at most $2n - 1$ patterns, not containing any gaps.

For the inductive step, an $n + 1$ -th rule needs to be added to the table, such that $r = n + 1$. The following will demonstrate that no matter how this $n + 1$ -th rule is added, the resulting segmentation will not contain more than $2(n + 1) - 1 = 2n + 1$ unique segments.

As a first option consider the situation, where the new rule does not overlap with any of the other rules. For example, assume without loss of generality that the rule is added to the right of all the other rules as depicted in Figure 4.13. This will only generate one additional segment (or pattern) such that now there are at most $2n - 1 + 1 \leq 2n + 1$ segments, so the formula is still valid.

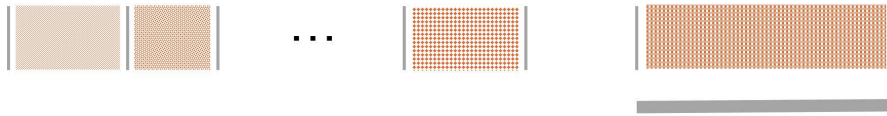


Figure 4.13: Induction step—Option 1

Next, assume the new rule is added such that it covers all of the existing segments and extends beyond them to the left and right.⁴ In this case, the existing patterns stay the same, because the same change is made to every one of them, such that, if they were unique before (which is the assumption), they will be unique afterwards. Therefore, after adding the new rule, only one additional pattern is introduced, namely to the left and right of the existing ones. This is illustrated in Figure 4.14. Hence, there are now at most $2n - 1 + 1 \leq 2n + 1$ unique segments, so the formula is still valid. Note as a special case of this situation, that if the rule does not extend to at least one side of the existing segments, not even a single new unique segment would be added.



Figure 4.14: Induction step—Option 2

As a third option, the rule is added such that it cuts (i. e., overlaps with but does not cover) one segment and covers the other segments that lie on the way to that cut point. As in the previous situation, the segments that are covered do not change their patterns. The segment that is cut keeps its pattern where it does not overlap with the new rule, and changes its pattern where it does overlap. An example is shown in Figure 4.15. The added rule cuts the rightmost segment and extends to the right of it, which leads to two new patterns. Note

⁴ For example, this could be a rule that simply matches any input.

as a special case of this situation, that if the rule does not extend to the right (or left) of the existing segments, only a single new unique segment would be added. Therefore, in this situation there are at most $2n - 1 + 2 \leq 2n + 1$ unique segments, so the formula is still valid.

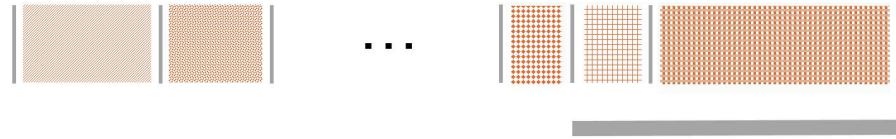


Figure 4.15: Induction step—Option 3

Lastly, the rule could be added so that it cuts two existing segments, one on each of its endpoints. Again, the segments that lie between these endpoints are covered by the new rule so that they do not change their patterns. Moreover, the segments that are cut will each keep the existing pattern and add a unique new pattern, as was the case in the previous situation when only one segment was cut. This is illustrated in Figure 4.16. Therefore, for this option there are at most $2n - 1 + 2 \leq 2n + 1$ unique segments, so the formula is still valid.

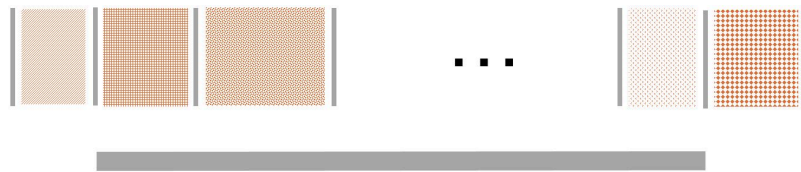


Figure 4.16: Induction step—Option 4

Besides these four options (including their special cases), there is no other way of adding another rule to a table with $r = n$ rules. Therefore, the hyperrectangles of the resulting table can only be partitioned into at most $2(n + 1) - 1 = 2n + 1$ unique segments. This completes the proof that the hyperrectangles of a decision table with r rules and one input column can be partitioned into at most $2r - 1$ unique segments. \square

Corollary 4.4.2.1. The least upper bound of the number of distinct outputs of a multi-hit DMN decision table with r rules and one input column is $2r - 1$.

Note that this bound only holds for multi-hit tables with one input column, since they have the ability to return combinations of output values. If a decision table can only return a single value, i. e., if it is a single-hit table, the least upper bound of the number of distinct outputs a table with any number of input columns is given by the number of rules r .

Regarding the question that was raised in the beginning of this section, Corollary 4.4.2.1 only provides an answer for the case of one

input column. It remains to determine the least upper bound of distinct outputs for tables with more than one input column. However, solving this problem requires a very intricate analysis of the different ways in which a set of n -dimensional hyperrectangles can overlap and is beyond the scope of this thesis.

FORMALIZING DECISION-AWARE PROCESS MODELS

Decision-aware business process models are characterized by their integration of two types of models, process models and decision models. Section 4.3 explained that decision models, whose behavior is defined by decision tables, can be viewed as conventional functions, mapping input values to output values. Consequently, decisions are simply operations on process data [107]. As already mentioned in Chapter 2, Petri nets can be used as a means to precisely express the semantics of business process models, yet they do not consider data. Therefore, colored Petri nets are required to incorporate the semantics of decision models in the formalization of a decision-aware business process model.

In Section 5.1 and Section 5.2 mappings from decision-aware process models to colored Petri nets are defined. On the basis of such a mapping, the concepts of colored workflow nets and their soundness are defined in Section 5.3, while Section 5.4 explains how to deal with certain cases of concurrent decision executions in the mapping. Finally, Section 5.5 summarizes the chapter with a short discussion.

5.1 SIMPLE COLORED PETRI NET MAPPING

Section 2.1.2.1 described a mapping of BPMN process models to Petri nets, abstracting from data. This mapping can be used as a basis for the mapping of decision-aware process models to colored Petri nets. The major difference is the consideration of decision models and their data operations in the mapping. Indeed, the mapping needs to be extended to consider the behavior of the process' *decision fragments* (cf. Definition 20). This includes the representation of the decision logic in the colored Petri net as well as the logic of the succeeding gateway.

5.1.1 *Merging Decision Tables*

The mapping is based on the assumption that the logic of the decision models is expressed using decision tables. Furthermore, if the model contains more than one decision, i. e., one top-level decision (returning the final decision value), plus one or more sub-decisions, the model first needs to be preprocessed to express the entire decision logic in a single (top-level) decision table, whose logic can then be represented in the colored Petri net. Since structuring decision models

into sub-decisions can be seen as syntactic sugar, this preprocessing step is straightforward and will be explained using an example.

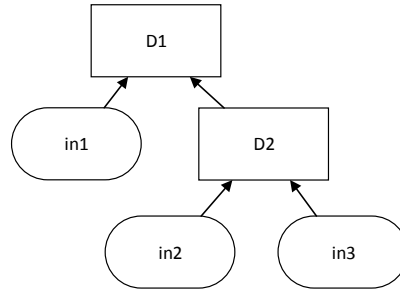


Figure 5.1: Decision requirements diagram with more than one decision

U	in1 Number	D2 Boolean	D1 {10%, 5%, 0%}
1	≥ 5	T	10%
2	< 5	T	5%
3	-	F	0%

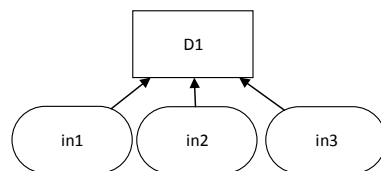
(a) Decision table for D1

U	in2 {A, B, C}	in3 Number	D2 Boolean
1	A	-	T
2	B	0	T
3	C	≠ 0	F

(b) Decision table for D2

Figure 5.2: Decision tables associated with the decisions in Figure 5.1

Figure 5.1 shows a decision diagram of a decision model with two decisions, *D1* and *D2*, where the output of *D2* serves as an input for *D1*. The respective decision tables are depicted in Figure 5.2. To merge the two tables into one, each condition of table *D1* that corresponds to an output value of *D2* needs to be replaced by the input conditions of *D2* that lead to this output. The resulting table is shown in Figure 5.3b. Instead of having *D2* as an input, this table receives the inputs of *D2*, namely *in2* and *in3*. Moreover, the input conditions that were previously imposed on the former input *D2* are now replaced by the input conditions on *in2* and *in3* that led to the respective values for *D2*. For example, rule one of the table in Figure 5.2a requires *D2* to be true. So now, instead of having *D2* as an input, this table receives the inputs of *D2*, *in2* and *in3*, and rule one is replaced by two new rules because *D2* can be true in two different situations (cf. Figure 5.2b).



(a) Merged DRD

U	in1 Number	in2 {A, B, C}	in3 Number	D1 {10%, 5%, 0%}
1	≥ 5	A	-	10%
2	≥ 5	B	0	10%
3	< 5	A	-	5%
4	< 5	B	0	5%
5	-	B	≠ 0	0%

(b) Merged decision table

Figure 5.3: Equivalent decision model with merged decision tables

5.1.2 Mapping Decision Fragments to CPNs

As explained in the beginning of this section, to express the behavior of a decision fragment in a colored Petri net, the decision logic referenced by the decision task as well as the logic of the succeeding gateway need to be mapped.

The mapping of the decision logic (expressed as decision tables) requires all decision tables of the decision model to be merged (as described in the previous section), and it requires the single merged table to be uniqueified using the method described in Section 4.2. This ensures that all rules are exclusive, such that each rule of the table can simply be mapped to a transition in the colored Petri net.

The transition guard represents the conditions of the rule as a conjunctive Boolean expression, and the transition’s outgoing arc expression represents the output of the rule. If a rule condition is empty (“–”), the corresponding conjunct of the guard will simply be omitted. Moreover, each input/output variable of the table is represented by a place that is connected to each transition. To represent the execution of the decision task that references the decision table, two control flow places are added, which the transitions read from and write to, respectively.

Concerning the color sets, the places representing the input/output variables of the table are assigned a color set that has the same domain as the corresponding table variable. The color set of the control flow places is the default *Unit* color set containing the single value “()”.¹

For example, suppose that a decision task of a process references the decision table in Figure 5.4a. Then, the decision task together with the decision table can be mapped to the colored Petri net in Figure 5.4b.

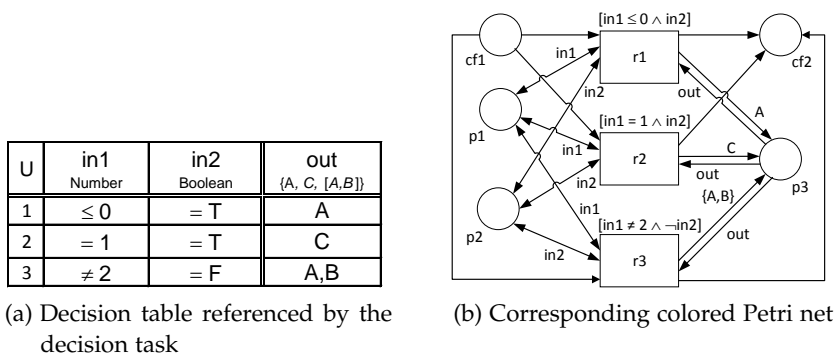


Figure 5.4: Example of a mapping of a decision task that references the decision table in 5.4a to a colored Petri net

¹ For this reason, the arc expressions of the incoming and outgoing arcs of a control flow place will always be “()”, and will therefore be omitted.

In this net:

- $\Sigma = \{Int, Bool, \{A, C, \{A, B\}\} \cup \emptyset, Unit\}$, where $Unit = \{()\}$,
- $V = \{in1 : Int, in2 : Bool, out : \{A, C, \{A, B\}\} \cup \emptyset\}$,
- $col(p1) = Int, col(p2) = Bool, col(p3) = \{A, C, \{A, B\}\} \cup \emptyset,$
 $col(cf1) = col(cf2) = Unit.$

Note that each transition reads *and* writes from and to the places representing the decision table variables. Regarding the places representing the input variables, this is to not consume the contained tokens “away”, as they may be needed again later on by other transitions using the same inputs. Regarding the places representing the output variables, this is to ensure that the place is 1-bounded, which would not be the case if the table was part of a loop and the read arc was missing.

In addition to mapping the decision task and its associated decision table, also the succeeding gateway must be mapped. This mapping works similarly by representing each outgoing edge of the gateway with a transition whose guard corresponds to the condition assigned to the edge. In conclusion, the steps explained above provide a way to map a decision fragment to a colored Petri net.

5.2 ABSTRACT COLORED PETRI NET MAPPING

The mapping of the previous section faces a major problem that will be described and overcome in this section.

5.2.1 Symbolic Abstraction

The problem with the mapping in Figure 5.4 is that it will lead to a reachability graph with infinitely many states. This is due to the fact that there are infinitely many initial markings for the net in Figure 5.4b, since the variable *in1* can take on infinitely many values, and if, for instance, soundness is to be checked, every possible value needs to be checked.

A solution to this type of problem is given by the idea of symbolic abstraction [17], where the general idea is to represent infinitely many values by an abstract symbol. In our case, the idea is as follows: Although there are infinitely many combinations of values for the tokens in *p1* and *p2* in Figure 5.4b, through the guard conditions of the net, those values will be *partitioned into finitely many sets of values*. And each partition will result in the same trace, i. e., its elements will enable and fire the exact same transitions when flowing through the net.

For example, for the net in Figure 5.4b there will be four partitions of the infinite set $\mathbb{Z} \times \mathbb{B}$:

- all the values in $\{z \in \mathbb{Z} \mid z \leq 0\} \times \{\top\}$ can only enable and fire transition $r1$;
- all the values in $\{1\} \times \{\top\}$ can only enable and fire transition $r2$;
- all the values in $\{z \in \mathbb{Z} \mid z \neq 2\} \times \{\perp\}$ can only enable and fire transition $r3$;
- all the remaining values in $\mathbb{Z} \times \mathbb{B}$ cannot enable any transition (because the table in Figure 5.4a is incomplete).

Therefore, to conduct a reachability graph or soundness analysis, instead of working with concrete values, we work with abstract symbols given by the partitions of the originally infinite set of values. Hence, rather than representing concrete values and operations on those values in the colored Petri net, sets are used and the operations like checking a condition of a decision table input, are changed to set operations.

For example, the abstract symbol for all the values of the variable $in1$ of type integer is given by the set \mathbb{Z} . Similarly, the values of $in2$ of type Boolean are represented by \mathbb{B} . The variables and the transition guards need to be updated accordingly. For example, the guard of transition $r1$ then is

$$[in1 \cap \{z \in \mathbb{Z} \mid z \leq 0\} \neq \emptyset \wedge in2 \cap \{\top\} \neq \emptyset],$$

using the intersection operator to check the abstract symbols for values that satisfy the respective conditions.

Similarly, the transition guard of $r3$ is

$$[in1 \setminus \{2\} \neq \emptyset \wedge in2 \cap \{\perp\} \neq \emptyset],$$

where in the first conjunct the complement operator is used to check the abstract symbol for values that satisfy the condition $\neq 2$.

Once a transition has fired, the symbols can be refined by the arc expressions of the arcs leading from the transition back to the variable places. For instance, if transition $r1$ fires, only the subset of values that actually fulfill $r1$'s conditions are written back. This expresses the fact that $r1$ creates the first partition of the list of partitions above. For example, the arc expression of the arc from transition $r1$ to place $p1$ is

$$in1 \cap \{z \in \mathbb{Z} \mid z \leq 0\}.$$

Clearly, when starting with the sets \mathbb{Z} and \mathbb{B} in the beginning, all transitions are enabled, and only after firing one of them, a subset of each set is created. Therefore, in the reachability graph a split will be encountered, where each outgoing edge corresponds to the firing of a

transition, and each path in the graph analyzes one of the partitions. This will be illustrated in the following.

Figure 5.5 shows the mapping of a decision task that references the table in Figure 5.4a, employing symbolic abstraction. It contains the two control flow places $cf1$ and $cf2$, and three variable places. Moreover, each rule of the decision table is represented by a transition.

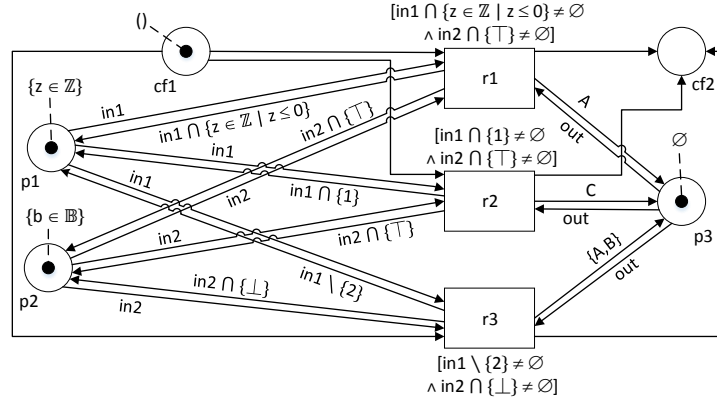


Figure 5.5: Colored Petri net corresponding to the table in Figure 5.4a, employing symbolic abstraction

The further specifications of this net are as follows:

- $\Sigma = \{2^{Int}, 2^{Bool}, \{A, C, \{A, B\}\} \cup \emptyset, Unit\}$, where $Unit = \{\{\}\}$,
- $V = \{in1 : 2^{Int}, in2 : 2^{Bool}, out : \{A, C, \{A, B\}\} \cup \emptyset\}$,
- $col(p1) = 2^{Int}$, $col(p2) = 2^{Bool}$, $col(p3) = \{A, C, \{A, B\}\} \cup \emptyset$,
 $col(cf1) = col(cf2) = Unit$.

The color sets in Σ were modified so that the input variable places are now defined as powersets of the original domains to enable the representation of sets of values instead of single values only. Accordingly, also the variables in V and the color sets of the places were adapted.

Comparing the nets in Figure 5.4b and Figure 5.5 it should be noted that they do not differ structurally, i. e., they contain the same places, transitions, and arcs between them. However, the color sets are defined differently, and therefore also the operations on the tokens of the respective color sets have been adapted, as can be seen from the guards and arc expressions, which now work on sets.

Also, the net has been given an initial marking. The markings of the places representing decision table inputs are given by the entire domains of the respective variables, acting as the corresponding abstract symbols as discussed above. The marking of the output variable place is the empty set, and the marking of the first control flow place is " $\{\}$ ".

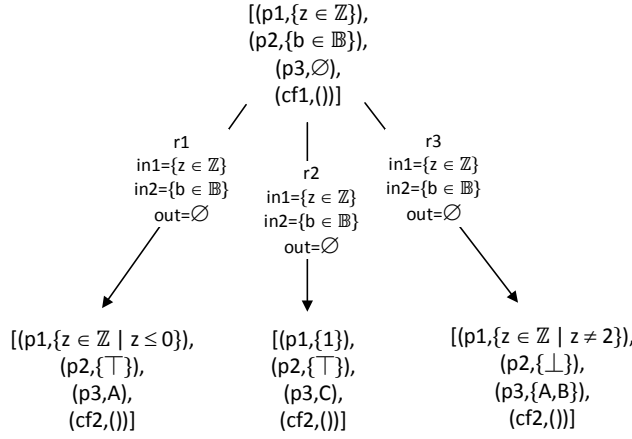


Figure 5.6: Reachability graph of the colored Petri net in Figure 5.5

Figure 5.6 shows the reachability graph of the colored Petri net in Figure 5.5. This graph shows how the Cartesian product of the input variable sets given by the initial marking is partitioned such that each partition represents the execution of all elements contained in that partition. Namely, three partitions reach a marking containing a token in place $cf2$, while the remaining partition deadlocks at $cf1$ due to the incompleteness of the decision table.

Those partitions are exactly the ones that have been described in the list above, and they are reached by the same transitions as above. Therefore, the nets in Figure 5.4b and Figure 5.5 exhibit the same behavior for the same data values. This in turn means that any behavioral analysis that needs to be conducted for the net in Figure 5.4b can be conducted on the net in Figure 5.5, and any property that is true for the latter will be true for the former.

5.2.2 Mapping Decision Fragments to Abstract CPNs

This leads to the generic mapping of a decision fragment to a colored Petri net. The generic decision fragment is shown in Figure 5.7. The decision task is associated with a decision model implemented by the decision table in Figure 5.7b. Without loss of generality, we assume that this table has two rules, two inputs, and one output. The two inputs are contained in the data object In read by the decision task, and the output is written into the data object Out written by the task. The input $in1$ is assumed to be an input variable of the process, while $in2$ is a variable that was written by a previous decision of the process.

Furthermore, the rule conditions compare the input variables with some value or list of values (such as an interval) of the variables' domains. The allowed operators are equality ($=$) and containment (\in), and their negations (\neq , \notin), all of which are used in the table in

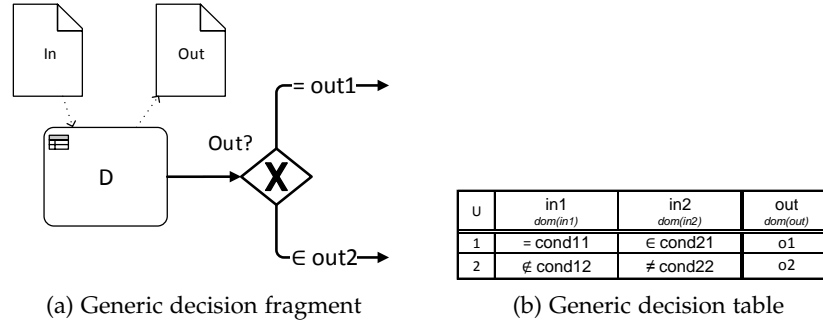


Figure 5.7: A generic decision fragment with its associated decision table

Figure 5.7b. Note that comparison operators such as “ \leq ” are special cases of the containment operator.

Finally, the table must have a unique hit policy. If this is not the case, it can be uniqueified using the procedure described in Section 4.2. For this reason, $dom(out)$ is assumed to be a set of multisets as defined in Definition 25. For simplicity, multisets with a single element, such as $\{A\}$ are represented as the value A . Hence, the outputs $o1$ and $o2$ are (totally ordered) multisets from $dom(out)$, if they contain more than one element. Otherwise, they are single values.

Without loss of generality, we assume that the split gateway succeeding the decision task has two outgoing edges. The edges are associated with conditions, where the condition $= out1$ tests the decision output for equality with the value $out1$, while the other condition checks if the output is contained in the set $out2$.

In general, the allowed edge condition operators are equality ($=$), containment (\in) and reverse containment (\ni), and their negations (\neq , \notin , \nexists).² Equally to decision table outputs, the edge condition value is always assumed to be a multiset, or a single value. For example, $out1$ could be the multiset $\{A, A, C\}$, but it could also be simply the value A .

The mapping is shown in Figure 5.8. Those parts of the table rules that refer to abstract symbols (i. e., $in1$) are mapped in such a way that they check the abstract symbols (i. e., sets) of the input values for an intersection with or a complement of the values corresponding to the respective input conditions. The complement operator is used in case of negated input conditions (through the operators \neq or \notin), while the intersection operator is used for the others.

For the other parts of the rules (referring to $in2$) the original operations are mapped—as it was done for the mapping in Figure 5.4b—because outputs of previous decisions are *concrete* values. Therefore, the guard of a decision table transition may implement symbolic and non-symbolic operations at the same time.

In order to enable a transition, the intersection or complement must not be the empty set in case of symbolic inputs, while for concrete

² If the operator in a table or edge condition is omitted, it defaults to $=$.

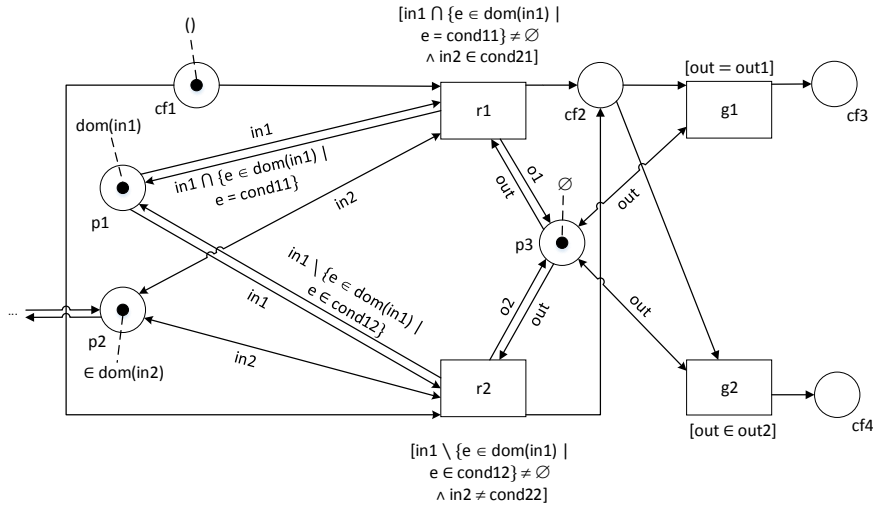


Figure 5.8: Colored Petri net corresponding to the decision fragment in Figure 5.7, employing symbolic abstraction

inputs the respective condition must simply be true. If a transition fires, it not only produces the respective output, but also refines the abstract symbols and writes them back to the input places.

Additionally, the gateway with its outgoing edges is mapped. For each outgoing edge a transition is created and the edge's condition is mapped to the guard of the transition.

As can be seen in the figure, the guard conditions also implement the original operations of the edge conditions in the process model. This is because they always read from an output variable place, which contains a concrete decision table output. Therefore, the edge condition operators can be mapped as they are, meaning the equality operator is mapped to an equality operator in the CPN, while the containment operators are mapped to containment operators in the respective directions (\in , \ni).

So, in general, the arc expressions and parts of the guards concerned with bindings from input variable places will implement symbolic operations and refinements of the symbols, whereas bindings from output variable places are treated as in Figure 5.4b (and the gateway transitions in Figure 5.5).

Note that a variable can be used by multiple decision tables in the process, such as the variable represented by p_2 in Figure 5.5. In that case, this is because the output of one table is used as an input for another table. Another possibility would be that two decision tables share the same input variables. However, this will not lead to multiple places for the same variable. Rather, for each variable exactly one place is generated that any transition can read from and write to.

In the end, this all boils down to the fact that each data object attribute referenced by the decision tables is mapped to exactly one place in the colored Petri net.

The further specifications of this net are as follows:

- $\Sigma = \{2^{dom(in_1)}, dom(in_2) \cup \emptyset, dom(out) \cup \emptyset, Unit\}$,
where $Unit = \{()\}$,
- $V = \{in_1 : 2^{dom(in_1)}, in_2 : dom(in_2) \cup \emptyset, out : dom(out) \cup \emptyset\}$,
- $col(p_1) = 2^{dom(in_1)}, col(p_2) = dom(in_2) \cup \emptyset, col(p_3) = dom(out) \cup \emptyset, col(cf_1) = col(cf_2) = col(cf_3) = col(cf_4) = Unit$.

This mapping can be embedded into the usual mapping of BPMN process diagrams to Petri nets: whenever a decision fragment is encountered the mapping in Figure 5.8 is applied, otherwise the mapping rules in Figure 2.8 are used, and the *Unit* color set is assigned to all control flow places.

5.2.3 Assumptions

The described mapping comes with the following assumptions:

- *input variables of decision tables are written as data object attributes by the start event or by a decision task*: no other tasks can write these attributes because their input-output behavior is unknown—unlike that of decision tasks;
- *input variables written by the start event cannot be overwritten*: such data represents input data to the process, and to properly verify the soundness of a colored Petri net, this input data must not be overwritten (cf. criterion (v) of Definition 28 below);
- *all data objects read by the decision tasks are either written by the start event or by a previously executed decision task*: this is to guarantee that when a data object is read, it was written before, so that it contains a value;
- *split and join gateways are either of type xor or and*: inclusive gateways are not allowed because of the difficulty of formalizing them [72];
- *xor split gateways only occur in decision fragments*: xor split gateways react on the outcome of decisions and should therefore be preceded by decision tasks that explicitly implement the decision logic;
- *decision tasks that are executed concurrently do not write the same data object*: this is to ensure the independence of the concurrent branches and will be further discussed in Section 5.4;

- the data types of the variables used in decision tables and edge conditions are Boolean, String, number (i. e., real numbers and all of their subsets such as integers), as well as enumerations of any sort of objects;
- the decision table conditions and the edge conditions are always of the form “*attribute operator constant*”, where *attribute* is the variable name, *operator* is an admissible operator for the respective data type, and *constant* is a concrete value from the domain of the data type. For example, in case of a number data type a valid condition would be $x < 5$. At the same time the output of a decision table is always a concrete value from its codomain.

5.3 COLORED WORKFLOW NETS AND THEIR SOUNDNESS

Having translated decision fragments to colored Petri nets, it is possible to define the notion of a *colored* workflow net, similar to the notion of a workflow net introduced in Definition 4. This definition imposed certain requirements on the structure of the Petri net. The same requirements are imposed on colored Petri nets. However, in such nets there are not only places representing the current state of the control flow of the underlying business process, but also places representing the current state of its variables. Therefore, these two sets of places need to be treated differently.

Regarding the control flow places, there still must be exactly one place with no incoming arc, and exactly one place with no outgoing arc, and every place and transition must be on a path between these two dedicated places. With respect to the variable places, they must all be connected to at least one transition, and if they are connected to a transition, that transition must both read from and write to the variable place. This is to ensure that a variable place will hold exactly one token at any point in time (given that it holds a token in the initial marking). Also, there are requirements on the color sets of the control flow and variable places.

For the data objects DO of a process model m and the places P of the colored Petri net CPN representing m , we define the function $Pl : DO \rightarrow 2^P$, such that for all $do \in DO$ the places representing the relevant attributes of do are given by $Pl(do) \subset P$.³ Furthermore, let e_s be the start event of m and DF its data flow relation. Note that the places P of CPN can be partitioned into the set of control flow places P_{cf} and the set of variable places P_v . Then, $DO' = \{do \in DO \mid (e_s, do) \in DF\}$ is the set of data objects written by e_s , and $P_i = \{p \in P_v \mid \exists do \in DO' : p \in Pl(do)\}$ is the set of variable places of CPN , whose corresponding data object is written by the start event of m —the input variable places. Therefore, $P_o = P_v \setminus P_i$ is the set

³ This function is obtainable from the mapping rules described in the previous section.

of remaining variable places—the output variable places. Note that $P_i \cap P_o = \emptyset$. A colored workflow net is then defined as follows:

Definition 26 (Colored workflow net). Let $CPN = (P, T, A, \Sigma, V, col, grd, ex, init)$ be a colored Petri net. CPN is a colored workflow net if and only if:

- $P = P_{cf} \uplus (P_i \uplus P_o)$ (P consists of disjunct sets P_{cf} of control flow places and variable input and output places),
- $\forall p \in P_{cf} : col(p) = Unit$ (control flow places can only hold the value $()$),
- $\forall p \in P_i, \exists v \in V : col(p) = 2^{dom(v)}$ (input variable places can hold every subset of the domain of the process variable they represent),
- $\forall p \in P_o, \exists v \in V : col(p) = dom(v) \cup \emptyset$ (output variable places can hold every value of the domain of the process variable they represent, plus \emptyset),
- $(\exists i \in P_{cf}, \neg \exists t \in T : i \in t \bullet) \wedge \forall p \in P_{cf} : (\neg \exists t \in T : p \in t \bullet) \implies (p = i)$ (i is the initial place of CPN),
- $(\exists o \in P_{cf}, \neg \exists t \in T : o \in \bullet t) \wedge \forall p \in P_{cf} : (\neg \exists t \in T : p \in \bullet t) \implies (p = o)$ (o is the final place of CPN),
- $\forall p \in P_i \cup P_o : \exists t \in T : p \in \bullet t \wedge p \in t \bullet$ (every variable place is connected to at least one transition that reads from and writes to that place),
- the net $CPN' = (P, T \cup \{t'\}, A \cup \{(o, t'), (t', i)\})$ is strongly connected, i. e., every pair of nodes (places and transitions) of CPN' is connected via a directed path.

◇

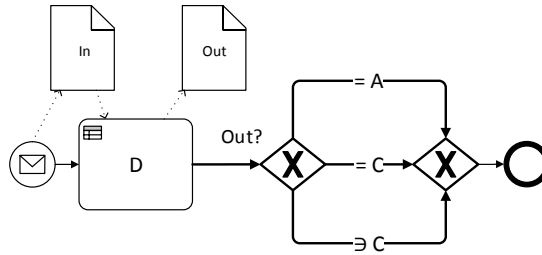


Figure 5.9: Decision-aware process model that references the table in Figure 5.4a and that can be translated to a colored workflow net

Consider the decision-aware business process in Figure 5.9. Using the mappings in Figure 2.8 and Figure 5.8, this model can be translated to a colored workflow net according to Definition 26.

Similar to a soundness analysis conducted for workflow nets, a soundness analysis for colored workflow nets can be performed. For that an initial marking M_I and possible final markings $M_f \in M_F$ are required that represent the initial state and possible final states of the business process. Note that there is not only a single final state (as it is the case for workflow nets), but potentially several. This is because the marking of a colored Petri net is not only defined by the distribution of tokens across the net, but also by the values of those tokens. And since a decision-aware business process may terminate with different values for its variables, there can be multiple final markings. The initial marking is characterized as listed below:

- The initial place of the net contains a *Unit* token;
- if the attribute represented by a variable place is contained by a data object written by the start event, the place contains a token whose value is the entire domain of the attribute/variable;
- all other variable places contain a token whose value is the empty set;
- all other places are empty.

Hence, we define the initial and the final markings as follows:

Definition 27 (Initial and final markings of a colored workflow net). Let $CPN = (P_{cf} \uplus (P_i \uplus P_o), T, A, \Sigma, V, col, grd, ex, init)$ be a colored workflow net. Let $i \in P_{cf}$ be the initial place and $o \in P_{cf}$ be the final place of CPN . The initial marking M_I and the set of final markings M_F of CPN are given by:

- $M_I(i) = \{()\}$ (the single element of the *Unit* color set),
- $\forall p \in P_{cf} \setminus \{i\} : M_I(p) = \emptyset$,
- $\forall p \in P_i, \exists c \in col(p) : (\forall c' \in col(p) : |c| \leq |c'| \implies c = c') \wedge M_I(p) = c$ (c is the largest element of the color set of p , which contains the entire domain of the data object attribute represented by p),
- $\forall p \in P_o : M_I(p) = \emptyset$,
- $M_F = \{M_f \mid M_f(o) = \{()\} \wedge$
 $\forall p \in P_{cf} \setminus \{o\} : M_f(p) = \emptyset \wedge$
 $\forall p \in P_i \cup P_o : |M_f(p)| = 1\}$.

◇

Consequently, the soundness of a colored workflow net can be defined:

Definition 28 (Colored workflow net soundness). Let $CPN = (P_{cf} \uplus (P_i \uplus P_o), T, A, \Sigma, V, col, grd, ex, init)$ be a colored workflow net. Let $i \in P_{cf}$ be the initial place and $o \in P_{cf}$ be the final place of CPN and let M_I be its initial marking and M_F its set of final markings. Finally, let M be a marking and b a binding. CPN is sound if and only if:

- (i) $\forall M, \exists M_f \in M_F : (M_I \xrightarrow{*} M) \implies (M \xrightarrow{*} M_f),$
- (ii) $\forall M, \forall M_f \in M_F : (M_I \xrightarrow{*} M \wedge \forall p \in P_{cf} : |M(p)| \geq |M_f(p)|) \implies (M \in M_F),$
- (iii) $\forall t \in T, \exists M, M', b : M_I \xrightarrow{*} M \xrightarrow{(t,b)} M',$
- (iv) $\forall M : (M_I \xrightarrow{*} M) \implies (\forall p \in P_i \cup P_o : |M(p)| = 1),$
- (v) $\bigcup_{M_f \in M_F} \prod_{p \in P_i} M_f(p) = \prod_{p \in P_i} M_I(p).$

◇

Criteria (i), (ii) and (iii) are similar to the three criteria in Definition 13. Note that, regarding the third criterion about the liveness of transitions, in case of a colored workflow net, additionally a suitable binding b is required under which transition t can fire to change the marking M to another marking M' .

The fourth and fifth criterion in Definition 28, however, are new. Criterion (iv) requires that in any reachable marking there is exactly one token in each variable place. Criterion (v) requires that the union of the Cartesian products of the final markings of all places in P_i must be equal to the Cartesian product of the initial marking of all those places. This is because every partition of the Cartesian product of the abstract symbols held by the input variable places must end up in some final marking. Otherwise, if a partition does not reach a final marking, the values contained in that partition do not properly terminate, such that the net is not sound.

As an example, consider the process model in Figure 5.9. Its translation to a colored workflow net—shown in Figure 5.10—is an extension of the one in Figure 5.8, in that it additionally contains the full mapping of the decision fragment and also of the join gateway.

The net contains an initial marking according to Definition 27. This marking consists of abstract symbols for the places $p1$ and $p2$, the empty set for $p3$, and a *Unit* token for the initial place $cf1$. The markings of $p1$ and $p2$ are the entire domains of the respective variables, since the start event of the corresponding process model only writes those variables in the data object *In*. Consequently, the marking of $p3$ is simply the empty set.

Figure 5.11 depicts the net's reachability graph, whose root node is the initial marking. In this marking, the transitions $r1$, $r2$ and $r3$ are enabled under the same bindings, as illustrated by the outgoing edges of the reachability graph's first node. Assuming that in this

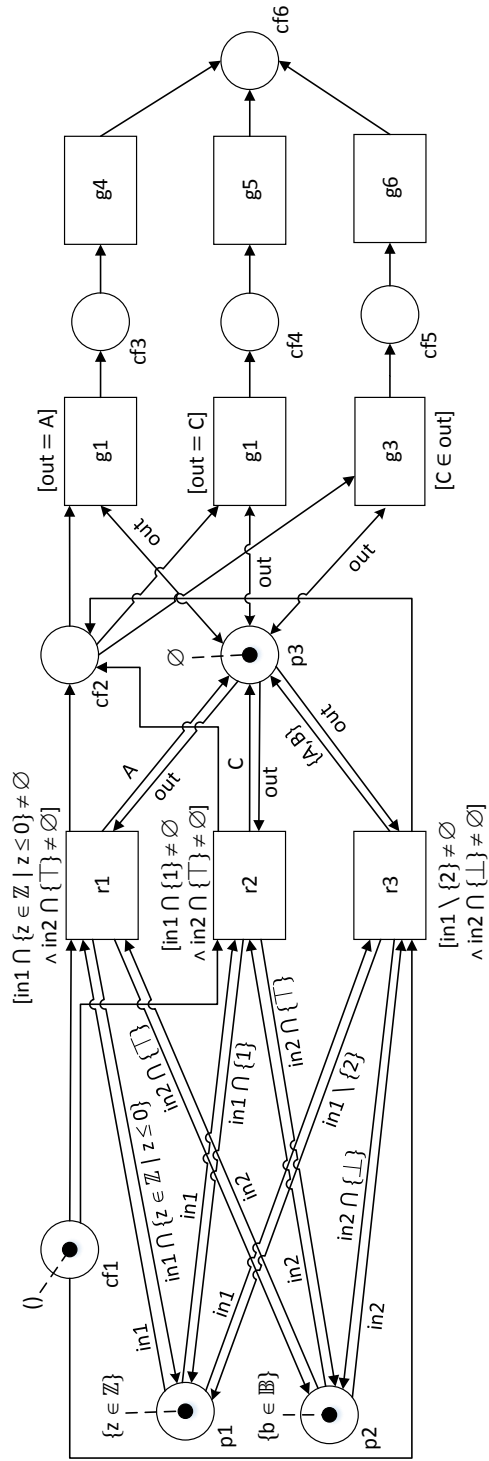


Figure 5.10: Colored Petri net mapping of the decision-aware process model in Figure 5.9

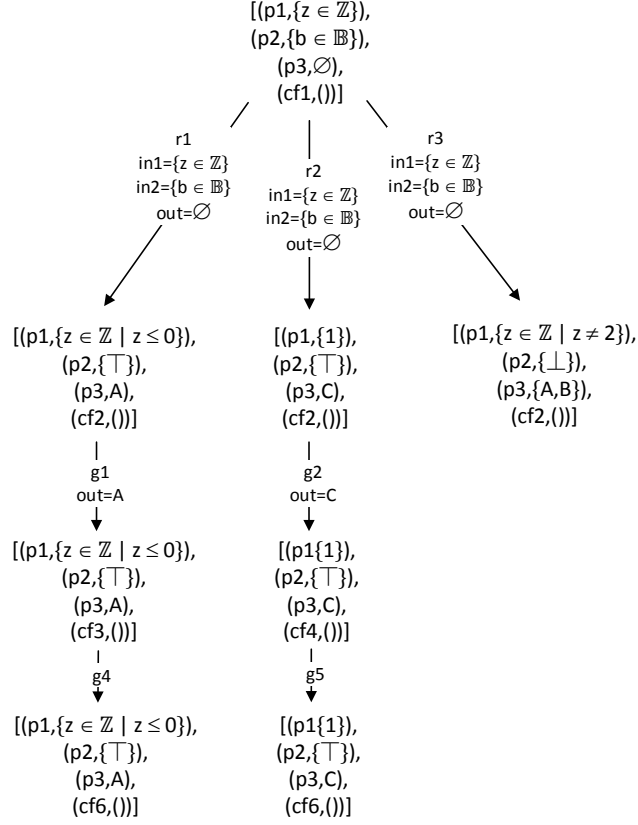


Figure 5.11: Reachability graph of colored workflow net in Figure 5.10

situation $r1$ is fired, a new marking is reached in which the tokens in places $p1$, $p2$ and $p3$ are updated according to the arc expressions of $r1$, and in which there now is a token in $cf2$ instead of $cf1$.

After firing $r1$, only transition $g1$ is enabled, whose firing leads to a marking in which the control flow token is moved from $cf2$ to $cf3$. Next, the join gateway transition $g4$ can be executed leading to a final marking.

The middle branch of the reachability graph behaves similarly. The rightmost branch, however, does not lead to a final marking. This is because no guard following the place $cf2$ is satisfied for the value $\{A, B\}$, such that the token in $cf2$ cannot continue. Hence, the net violates criterion (i) in Definition 28 and is not sound.

However, the net also violates criterion (v), because

$$\begin{aligned}
 \bigcup_{M_f \in M_F} \prod_{p \in P_i} M_f(p) &= (\{z \in \mathbb{Z} \mid z \leq 0\} \times \{\top\}) \cup (\{1\} \times \{\top\}) \\
 &= \{z \in \mathbb{Z} \mid z \leq 1\} \times \{\top\} \\
 &\neq \mathbb{Z} \times \mathbb{B} \\
 &= \prod_{p \in P_i} M_I(p).
 \end{aligned}$$

Apparently, not all possible input values of the process can properly terminate. The reachability graph shows that the values in $\{z \in \mathbb{Z} \mid z \neq 2\} \times \{\perp\}$ are stuck in the state with a control flow token in *cf2*, while all the remaining values are stuck in the initial state.

5.4 CONCURRENT EXECUTION OF DECISION FRAGMENTS

Coming back to the sixth assumption that was listed in Section 5.2.3 above for the mapping of decision-aware process models to abstract colored Petri nets—concerning the concurrent execution of decisions—we justify it using the example in Figure 5.12, which also shows a special case of that situation.

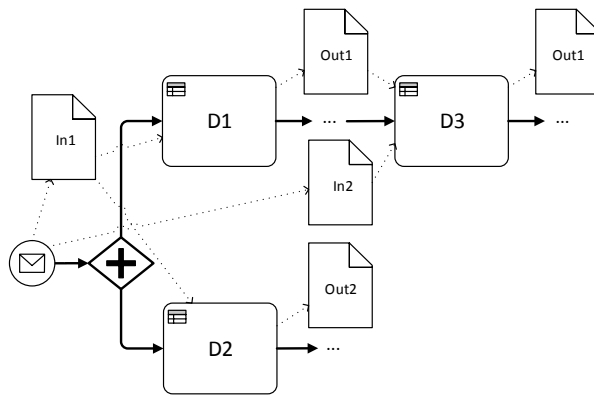


Figure 5.12: Decision tasks being executed concurrently

This figure shows a process model in which decisions are not only executed concurrently but also the concurrent decisions *D1* and *D2* reference the same decision model. Therefore, they read the same data object *In1*. Note, however, that they write different data objects, *Out1* and *Out2*, as required by the assumption above. This may seem unnecessary because they produce the same output value anyway, since they read from the same data object and reference the same decision logic.

The reason for this is clarified through the fact that on the upper branch there is another decision *D3*, that references another decision model but that has the same output variable as *D1* and therefore overwrites or updates *Out1*—which is admissible because *D1* and *D3* are executed sequentially. Now, if all of the decisions were writing the same data object, and *D2* was executed after *D3*, then the update made by *D3* would get lost and would not be available anymore in the upper branch. This certainly represents undesired behavior because it violates the independence of the concurrent branches.

Note that the requirement of writing different data objects must also be fulfilled if *D1* and *D2* reference different decision tables having the same output variable. The reason for assuming that they refer-

ence the same table in this example is to draw attention to the special treatment that this situation requires.

The fact that $D1$ and $D2$ are required to write different data objects implies that in the CPN there will be different output variable places, one representing $Out1$, the other $Out2$. This in turn means, however, that the transitions representing the decision table rules of $D1$ and $D2$ must be duplicated—even though the duplicates implement the same decision table rule. If both $D1$ and $D2$ were using the same transitions in the CPN, there would be no way of distinguishing for which of the two data object places the transitions should produce a token.

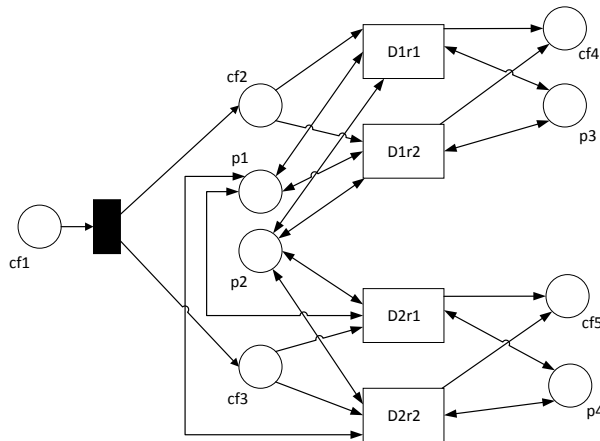


Figure 5.13: Partial mapping of the process model in Figure 5.12

The correct mapping of the two decisions $D1$ and $D2$ in Figure 5.12 is shown in Figure 5.13. In summary, decision fragments that are executed concurrently must write to different data objects, even if the decision tables have the same output variables. If two concurrent decision fragments reference the same decision table, each fragment is mapped independently according to Figure 5.8, with the exception that they will share the same input variable places. Hence, even though the tables are the same, for each of those tables another set of transitions (representing the table rules) will be generated (e.g., $D1r1$ in the figure). Also, they will use different output variable places ($p3$ and $p4$). But, they will use the same input variable places ($p1$ and $p2$). This represents the fact that they read from the same data objects of the process, while writing to different data objects.

5.5 SUMMARY AND DISCUSSION

This chapter demonstrated how decision-aware process models can be formalized such that their state space can be analyzed via a reachability graph. In principle, the reachability graph of a decision-aware process model contains infinitely many states, making an analysis of that graph—e. g., with respect to soundness—impossible.

However, we showed how the idea of symbolic abstraction can be used in this context to group the infinitely many states into finitely many partitions, such that the states of each partition display the same behavior regarding reachability.

It should be noted that this requires the decision logic of the DMN decision models to be implemented in the Simplified Friendly Enough Expression Language (**S-FEEL**) of the DMN standard [114], which is a simplification of DMN's **FEEL** language. S-FEEL only allows rule conditions of the form “attribute operator constant”, such as $x < 5$, and the rule outputs can only be constants. Therefore, outputs such as $x + 1$ are not allowed in S-FEEL. This would lead to problems regarding symbolic abstraction.

For example, consider a decision table that updates its input x by one, and that is part of a loop that is executed infinitely often (due to a modeling flaw). If an abstract symbol such as $\{x \in \mathbb{Z} \mid x \leq 2\}$ enters this loop it will be updated on each repetition, so that the reachability graph will contain infinitely many states—which is exactly the problem we aimed to solve. Hence, only DMN S-FEEL decisions are allowed. For a formalization of S-FEEL decisions the reader is referred to [97].

Part III

SOUNDNESS OF DECISION-AWARE BUSINESS
PROCESSES

STATELESS DECISION SOUNDNESS

This chapter introduces the notion of (stateless) decision soundness. The need for the definition of such a notion is motivated in Section 6.1. Therefore, that section also outlines the structure of the rest of this chapter. The work presented in this chapter has partly been published in [80].

6.1 MOTIVATION AND PROBLEM STATEMENT

Figure 6.1 shows a decision-aware BPMN process model that was already introduced in Section 2.1.2.4. Its two decision tasks are associated with the two decision tables displayed in Figure 6.2. Depending on the *BahnCardType* attribute of the *Booking* data object, the table in Figure 6.2a determines a discount which is then processed by the subsequent gateway and its outgoing branches. Next, a potential special offer is decided upon based on the previously calculated *Discount* and the *BahnCardType* (Figure 6.2b). Again, the result is processed by the following gateway and the process ends. Hence, the process model essentially consists of two decision fragments.

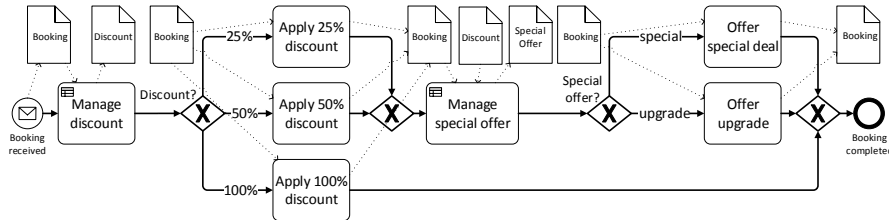


Figure 6.1: Train ticket booking process derived from booking tickets with Deutsche Bahn, modeled as a BPMN process diagram

U	BahnCardType {25, 50, 100}	Discount {25%, 50%, 100%}
1	25	25%
2	50	50%
3	100	100%

(a) *Manage discount* decision table

F	BahnCardType {25, 50, 100}	Discount {25%, 50%, 100%}	Special offer {none, upgrade, special}
1	25	25%	special
2	50	-	upgrade
3	-	100%	none

(b) *Manage special offer* decision table

Figure 6.2: Decision tables referenced by the decision tasks of the process model in Figure 6.1

Analyzing this process model for classical soundness according to Definition 13 would lead to the conclusion that this is actually a sound process model. This is because it can be translated to a workflow net and the corresponding reachability graph would show that

every execution will properly reach the final state and that every activity can participate in at least one execution. However, classical soundness only looks at control flow information and does not consider the data that determines under which conditions which branches and activities of the process can be executed.

This, however, is an important aspect of decision-aware processes. For example, the three outgoing branches of the first decision fragment are associated with conditions whose values are set by the preceding decision task which in turn relies on the decision table in Figure 6.2a. If this table can produce values that are different from the values expected at the branches of the split gateway, the process may deadlock. An example of this situation is given by the second decision fragment. The third rule of the table associated with this fragment produces the output value *none*. But this value does not occur in any of the branch conditions of the corresponding decision fragment. Hence, an execution in which the table returns *none* will deadlock because none of the branches can be selected to continue the process. Consequently, the process model cannot be considered sound.

Therefore, in the remainder of Part iii of this thesis, criteria and techniques are discussed that enable the correct verification of the soundness of decision-aware process models such as the one in Figure 6.1. The remainder of this chapter focuses on criteria that provide a simple way to ascertain the soundness of decision fragments in isolation. That means that the criteria are checked locally for each decision fragment of the process model. A structural check is described in Section 6.2, while behavioral checks are defined in Section 6.3. Altogether, this allows to define the notion of stateless decision soundness in Section 6.4. This approach has some limitations which are discussed at the end of this chapter in Section 6.5, and which are overcome in Chapter 7.

6.2 STRUCTURAL CONSISTENCY

A decision model, like a decision table, can be viewed as a function, that maps input values to output values. The input values are supplied by the process on which the decision performs some operations to produce an output value which is returned to the process. The interface between a process model (or decision fragment) and a decision model is therefore given by the process model's data objects, and the decision model's input data nodes and the output variable of its top-level decision. This interface must be consistent in the following manner:

Definition 29 (Structural consistency between decision fragment and decision model). A decision fragment is structurally consistent with its associated decision model if and only if:

- For each input data element in the decision model there is a data object in the decision fragment, such that
 - the decision fragment’s decision task reads this data object,
 - the input data element and the data object reference the same data model or the attributes of the input data element are contained in the attributes of the data object,
- the output produced by the decision model is written into a corresponding data object by the decision task,
- the outgoing edges of the decision fragment’s split gateway evaluate the output of the decision model via Boolean expressions.

◇

6.3 BEHAVIORAL CONSISTENCY

The notion of workflow soundness given in Definition 13 essentially consists of two aspects: proper termination (which includes the absence of deadlocks) and the absence of dead transitions. Both of these aspects can be affected by decision fragments, which is why in the following two decision soundness criteria will be introduced: *decision deadlock freedom* and *dead branch absence*. Since decision tables are standardized in DMN, we will illustrate the criteria using decision tables. A generalization will be discussed in Section 6.5.

6.3.1 Decision Deadlock Freedom

Decision deadlock freedom (DDF), as the name suggests, ensures that the integration of a process model with a decision model will not cause any deadlocks. In this regard, decision-induced deadlocks come in two flavors.

On the one hand, even if the process model and the decision model reference the same data model, the rules of the decision table reading the data object might not cover its entire domain. In this case, the process can supply inputs to the decision for which there is no matching rule in the responsible decision table. Hence, the decision table cannot be evaluated and will not return any output, causing the process to deadlock, thereby violating soundness criterion (i) (cf. Definition 13). Consequently, one aspect of DDF is the completeness of all decision tables of the decision model.

On the other hand, it must be ensured that the branch conditions of the decision fragment can handle all of the possible outputs of the decision model. If there is an output value for which there is not at least one matching branch condition, the process will again deadlock.

What does it mean for a table to be complete? The DMN standard states that a decision table is complete “if its rules cover all combina-

tions of expected input values". These input value combinations are given by $IV = \prod_{j=1}^n \text{dom}(i_j)$ in Definition 18, where IV contains all combinations of values from the domains of the input variables.

According to Definition 19, a decision table is associated with a decision node in a decision model. A decision node, in turn, is connected to other decision nodes or to input data nodes, via information requirement edges. Therefore, a table's input variable may be associated with the output value of another decision table or with the value of an input data element. Regarding the former case it is important to note that the set of possible outputs of a table may be a subset of the domain of the output variable of that table, as was discussed in detail in Chapter 4. This set of possible outputs can be easily determined based on the uniqueification method described in Section 4.2.

Table completeness as required for decision deadlock freedom is then defined as follows:

Definition 30 (Table completeness). A DMN decision table is complete if and only if

$$\forall iv \in IV, \exists ov \in OV, \exists r \in R : iv \times ov \in r.$$

◇

Consequently, for every possible input value combination iv , there must be an output value ov , and at least one rule r of the table covers this input-output relation.

Although having no incomplete decision tables in the decision model is a necessary condition to ensure decision deadlock freedom, it is not a sufficient one. It must additionally be ensured that all of the outputs of the decision model can be processed by the calling process. Since the outputs of a decision model are determined by its top-level decision, it follows that all possible outputs of the corresponding decision table must be *covered* by the process. "Covered" in this case means that for each output a branch condition associated with an outgoing edge of the decision fragment's gateway evaluates to true.

As an example, consider the decision fragment in Figure 6.3a. Given that the associated decision model produces, for instance, the output $\{x, y\}$, the upper branch condition—*output* = $\{x, y\}$ —evaluates to true for this output. The same is true for the upper branch condition in Figure 6.3b, which only requires the single element x to be present in the output.

This leads to the definition of the *decision deadlock freedom* (DDF) criterion, which is formulated for a decision fragment df with decision task a_D and edge conditions C_{df} . Furthermore, $dm = \delta(a_D)$ is the decision model associated with a_D , containing a set of decisions D and a corresponding set of decision tables DT . The top-level decision is called d_{top} , and $dt_{top} = \text{tab}(d_{top})$ is the decision table of the top-level decision.

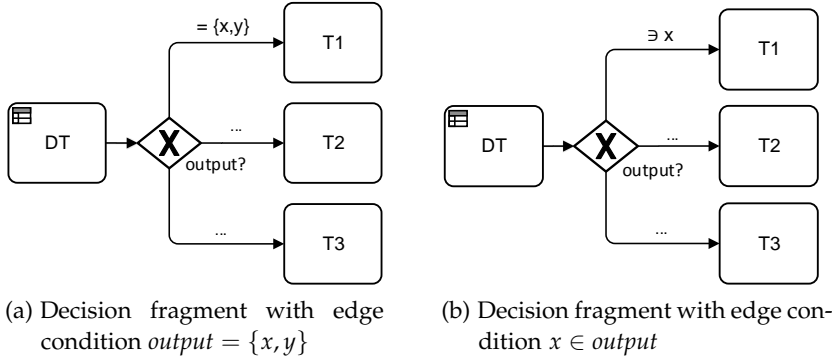


Figure 6.3: Different possibilities of covering table outputs by edge conditions

Definition 31 (Decision deadlock freedom). A decision fragment df satisfies the *decision deadlock freedom* criterion if and only if

- (i) $\forall dt \in DT : c(dt) = true$, (table completeness)
- (ii) $\forall out \in output(dt_{top}), \exists cond \in C_{df} : cond(out) = true$. (output coverage)

◇

Consider the first decision fragment in Figure 6.1, whose associated table is shown in Figure 6.2a. Since the table is complete and all of its outputs (25%, 50%, 100%) have matching conditions in the process model, this decision fragment is indeed decision deadlock free.

This, however, is not the case for the second decision fragment, referencing the decision table in Figure 6.2b. First, the table is incomplete because it does not contain a rule for the input (25, 50%). Should the process provide such a value combination as an input to the table, it would get stuck. Second, the third rule of the table produces an output value (*none*) that does not have any matching branch condition in the process model. Therefore, the output coverage part of the DDF criterion is violated as well and the second decision fragment of the process model is not decision deadlock free.

6.3.2 Dead Branch Absence

The second requirement for behavioral consistency between process and decision models is the absence of dead branches. This means that every branch condition of the decision fragment can be fulfilled by at least one output of the associated decision model, in order to follow that branch during execution. Otherwise, the branch whose condition can never be met will never be followed, and is therefore dead, which violates soundness criterion (iii) (cf. Definition 13). Hence, the *dead branch absence* (DBA) criterion is defined as follows:

Definition 32 (Dead branch absence). A decision fragment df satisfies the *dead branch absence* criterion if and only if

- (i) $\forall cond \in C_{df}, \exists out \in output(dt_{top}) : cond(out) = true$.

◇

Both decision fragments in Figure 6.1 satisfy the DBA criterion. Instead, consider the fragment in Figure 6.4a associated with the table in Figure 6.4b. The exclusive gateway of the process fragment has

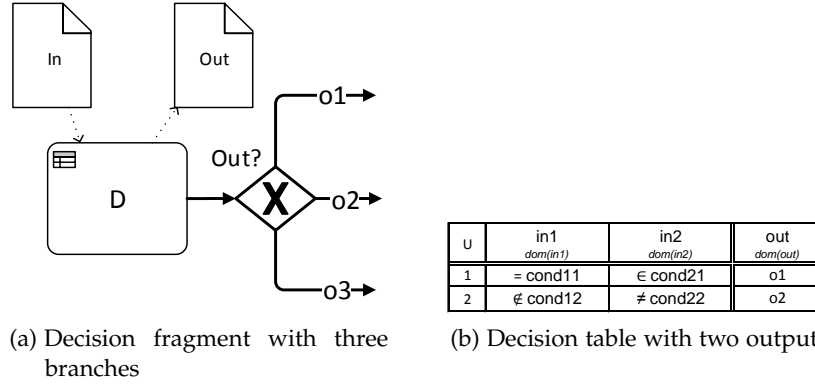


Figure 6.4: A decision fragment that violates the *dead branch absence* criterion

three outgoing branches with different conditions, one of which ($o3$) will never be satisfied by the decision table because that table will never produce such an output. Therefore, the corresponding branch is dead and the DBA criterion is violated.

6.4 STATELESS DECISION-AWARE SOUNDNESS

Having specified the structural and behavioral consistency between process models and decision models, it is possible to define the soundness of decision-aware process models. Note that the criteria described in Section 6.3 consider decision fragments in isolation. That is, they do not take into account the overall process into which they are embedded. This may be important because the process calls the decision model with some input data. For example, in Figure 6.4a the process calls the decision table with the current values of the data object In . These values determine which rule is matched in the table and which output will be produced correspondingly. Contrarily, the criteria formulated in Definitions 31 and 32 simply consider all possible outputs of the decision model (or table). This issue will be further discussed in Section 6.5 and dealt with in Chapter 7.

Hence, the following soundness definition is stateless because it does not consider the state of the process determining the possible inputs to the decision model call.

Definition 33 (Stateless decision-aware soundness). A decision-aware process model is stateless sound if and only if all of its decision fragments are

- (i) structurally consistent (Definition 29),
- (ii) decision deadlock free (Definition 31), and
- (iii) dead branch free (Definition 32).

◇

6.5 DISCUSSION AND SHORTCOMINGS

The definition of stateless decision-aware soundness assumes that the decision model that is embedded in the process model is called from within a decision fragment. Such fragments process the outcome of the decision via the branch conditions of a split gateway. However, this is not the only way a decision can be embedded in a process. As already described in Section 2.3 the situation could also be such as in Figure 6.5. In that case, the output of the decision (*Out*) is imme-

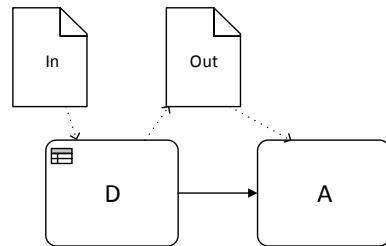


Figure 6.5: Decision task directly followed by another task reading the output of the decision

diately consumed by a task so that on model-level it never becomes observable if the process can actually respond properly to the decision outputs. The implementation of task *A* may or may not be able to handle all outputs. This is not observable on model-level and therefore the definition of decision-aware soundness is only suited for processes containing decision fragments as defined in Definition 20.

Finally, the shortcomings of Definition 33 are clearly given by the fact that it only considers decision fragments in isolation when checking soundness. This is especially worrisome given that decision models are supposed to be reusable [76]. Reusability entails that not every rule of the decision table may be relevant for a particular process model it is called from. Therefore, checking if every possible output of the decision table is met by a branch condition in the decision fragment is too strict and could designate decision-aware process models as unsound that do indeed not contain any unsound firing sequences. Hence, a definition that takes into account the context of the decision fragment would be based on considerations of which inputs can

be supplied to the decision model by the process model in question. Those inputs can be determined by analyzing the possible states of the business process at the time the decision called. Accordingly, in the next chapter *state-based* decision-aware soundness is examined.

STATE-BASED DECISION SOUNDNESS

This chapter builds on the notion of decision-aware soundness introduced in the preceding chapter. By recognizing the limitations described in Section 6.5, it explains how state space information can be used for a more sophisticated definition of decision soundness in Section 7.1. Subsequently, corresponding notions of state-based decision deadlock freedom and state-based dead branch absence are defined (Section 7.2 and Section 7.3). The work presented in this chapter has partly been published in [77].

7.1 USING STATE INFORMATION FOR SOUNDNESS CHECKING

Following up on the line of argumentation in Section 6.5, this section shows why state information is relevant when checking decision fragments for soundness. Consider again the decision-aware process model from the last chapter, repeated here for convenience together with its decision tables in Figure 7.1 and Figure 7.2.

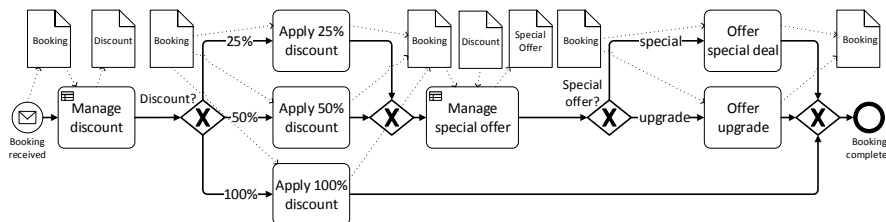


Figure 7.1: Train ticket booking process derived from booking tickets with Deutsche Bahn, repeated for convenience

U	BahnCardType (25, 50, 100)	Discount (25%, 50%, 100%)
1	25	25%
2	50	50%
3	100	100%

(a) *Manage discount* decision table

F	BahnCardType (25, 50, 100)	Discount (25%, 50%, 100%)	Special offer (none, upgrade, special)
1	25	25%	special
2	50	-	upgrade
3	-	100%	none

(b) *Manage special offer* decision table

Figure 7.2: Decision tables referenced by the decision tasks of the process model in Figure 7.1, repeated for convenience

While the first decision table is complete and its outputs perfectly match the branch conditions of the corresponding decision fragment, there seem to be some problems concerning the second decision fragment, as was already argued in Section 6.3.1: There is no rule for the input (25, 50%) and the output *none* is not covered by the branch conditions. However, looking at the context of the decision fragment,

it is noticeable that the task *Manage special offer* can only be reached if $Discount \in \{25\%, 50\%\}$, because those are the branch conditions of the preceding split gateway. From this, in turn, it can be deduced that the variable *BahnCardType* must either contain the value 25 or 50, since $Discount = 25\% \iff BahnCardType = 25$ and $Discount = 50\% \iff BahnCardType = 50$. This is apparent from the first two rules of the decision table in Figure 7.2a, and the fact that the values are not changed afterwards. Hence, the presumed problems of the second decision fragment regarding completeness and output coverage can never occur in this process.

Those considerations can be made more explicit by employing the formalization of decision-aware process models as colored Petri nets (CPNs) introduced in Chapter 5. This is because the CPN representation enables a state space (or reachability graph) analysis of the process at the point in time a decision is made.

Figure 7.3 shows a simplified version of the process model in Figure 7.1, omitting non-decision tasks in order to focus on the decision fragments. Using the mappings in Figure 2.8 and Figure 5.8 this pro-

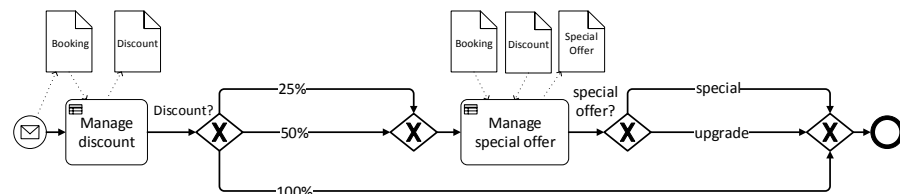


Figure 7.3: Simplified version of the process model in Figure 7.1

cess model is translated to the colored workflow net shown in Figure 7.4. The initial place of this colored workflow net is *cf1* in the top left corner of the figure, holding a token with value $()$.¹ Additionally, there are three variable places—*p1*, *p2*, *p3*—for the three variables used by the decision tables in Figure 7.1. *p1* initially holds a token with the entire domain of the respective variable since that variable is written in the *Booking* data object by the process’ start event, while the other variable places hold empty set tokens, as explained in Chapter 5.

Given this initial marking the net’s reachability graph can be constructed, shown in Figure 7.5. The root node of that graph is given by the initial marking. In that state, the decision task *Manage discount* with its three rules is enabled. All three transitions can fire under the same binding, namely $bct = \{25, 50, 100\}$ (representing the variable *BahnCardType*) and $disc = \emptyset$ (representing *Discount*). This is shown by the labels of the outgoing edges of the root node.

Inspecting the reachability graph further, it should be noted that only two of the three branches progress to states in which there is a

¹ Note that “()” was omitted from the arc expressions of the control flow places, for better readability.

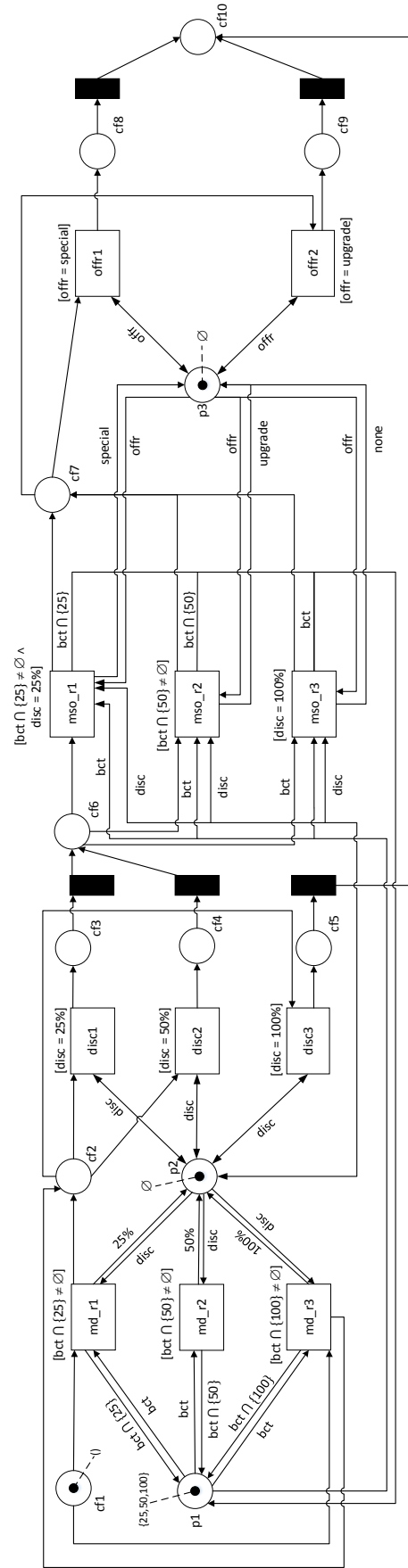


Figure 7.4: Colored Petri net mapping of the decision-aware process model in Figure 7.3

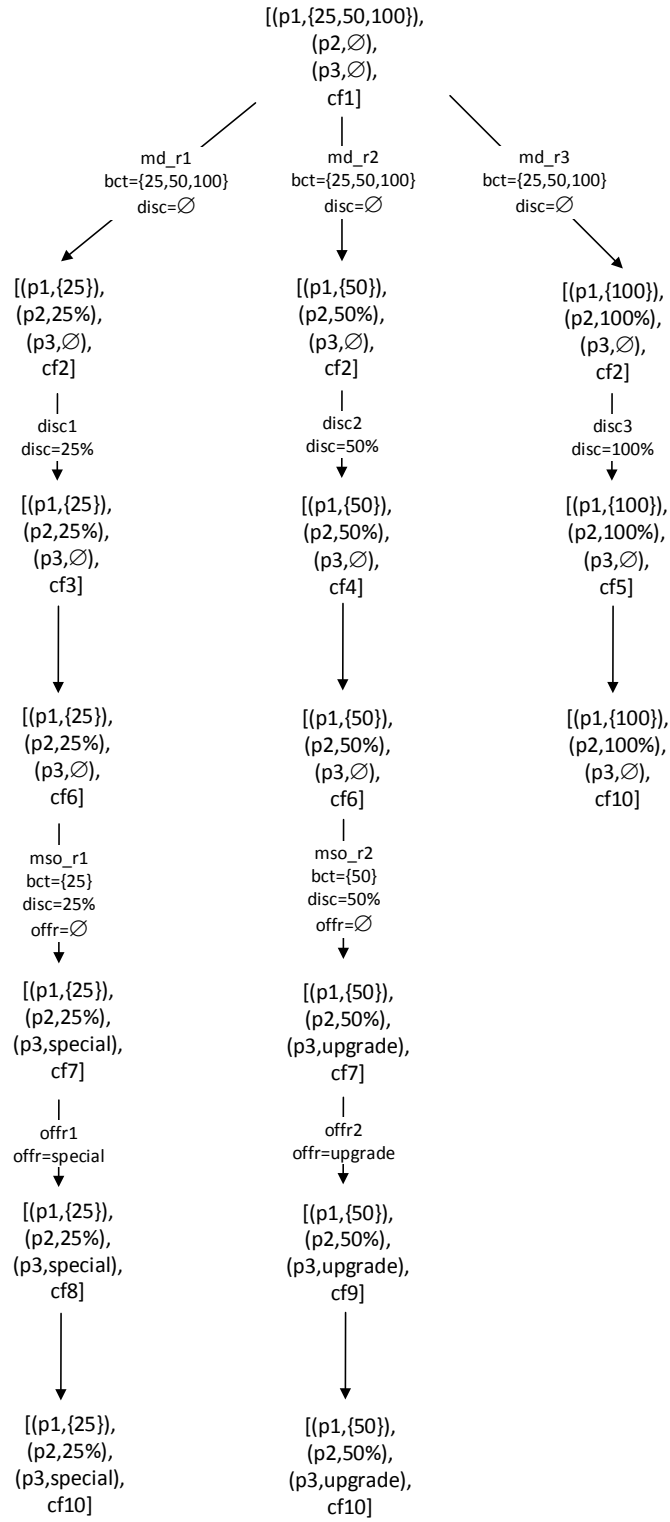


Figure 7.5: Reachability graph of the colored Petri net in Figure 7.4

control flow token in place $cf6$. In these states only the first two rules of the decision table *Manage special offer* can be executed, represented by the transitions mso_r1 and mso_r2 , with two different bindings. In case the process is currently in the state where $bct = \{25\}$ and $disc = 25\%$, mso_r1 can fire. This will set the $offr$ (representing the *Special offer* variable) to *special*, and put a control flow token in $cf7$. Analogously, progressing from the other state containing a token in $cf6$, $offr$ will be set to *upgrade*, and a token will appear in $cf7$.

Hence, there is no reachable marking that holds a token in $cf7$ and in which $offr$ is equal to *none*. Subsequently, all three branches correctly reach a final marking containing a token in the final place $cf10$ and in no other control flow place. Those considerations enable a *state-based* definition of the decision deadlock freedom and dead branch absence criteria.

7.2 STATE-BASED DECISION DEADLOCK FREEDOM

The first condition of decision deadlock freedom as specified in Definition 31 is that the decision tables of the decision model are complete, meaning that they cover all possible combinations of input values from the domains of the input variables. According to this requirement, the second decision fragment in Figure 7.3 would be unsound, because the table is missing the input combination (25, 50%). However, from state space analysis it becomes apparent that this input can never occur in this decision fragment. Therefore, the decision table is *conditionally complete*—conditioned on the possible states of the process at the point when the decision is made.

Similarly, the second condition of decision deadlock freedom is output coverage, which is violated by the decision fragment at hand because there is no branch condition matching the output *none*. Yet, the possible states after the decision was made do not include one in which the variable *Special offer* equals *none*. Hence, all outputs of the decision table are *conditionally covered*.

This leads to the definition of the *state-based decision deadlock freedom* (SB-DDF) criterion, which is formulated for a decision fragment df , that is part of a decision-aware process model. The decision fragment is represented by a colored Petri net fragment CPN_{df} , embedded into a colored workflow net CPN , representing the decision-aware process model. For convenience, let the initial place of CPN_{df} be given by i_{df} and its set of transitions by $T_{df} = T_{dt} \uplus T_g$, where $T_{dt} = \{t \in T_{df} \mid i_{df} \in \bullet t\}$ is the set of transitions representing the decision table rules of df , and $T_g = \{t \in T_{df} \mid \forall t' \in T_{dt} : \bullet t \setminus P_v = t' \bullet \setminus P_v\}$ is the set of transitions representing the gateway branch conditions. That is, T_{dt} are all the transitions that read from the first control flow place of df , and T_g are all the transitions that read from the second control flow place.

Definition 34 (State-based decision deadlock freedom). Let CPN_{df} be a colored Petri net fragment representing the decision fragment df , which is embedded into a colored workflow net CPN . Let i_{df} be the initial place of CPN_{df} and M_I be the initial marking of CPN . Further, let M be a marking and b a binding. The decision fragment df satisfies the *state-based decision deadlock freedom* criterion if and only if:

$$(i) \quad \forall M : (M_I \xrightarrow{*} M \wedge |M(i_{df})| > 0) \implies \exists t \in T_{dt}, M', b : M \xrightarrow{(t,b)} M' \text{ (conditional completeness),}$$

$$(ii) \quad \forall M, M', t \in T_{dt}, b : (M_I \xrightarrow{*} M \wedge |M(i_{df})| > 0 \wedge M \xrightarrow{(t,b)} M') \implies \exists t' \in T_g, b', M'' : M' \xrightarrow{(t',b')} M'' \text{ (conditional output coverage).}$$

◇

The first condition of SB-DDF says that, in any reachable marking in which there is a token in the initial place of CPN_{df} , there must be a decision table transition of CPN_{df} that can fire from that marking.

The second condition says that, in any marking that is reachable by firing any of the decision table transitions of CPN_{df} , there must be a gateway transition of CPN_{df} that can fire from that marking.

Figure 7.6 shows a sketch of a CPN decision fragment CPN_{df} to visualize those conditions. In this example, $T_{dt} = \{T_{dt1}, T_{dt2}\}$ and $T_g = \{T_{g1}, T_{g2}\}$. The net is in a state in which the initial place of CPN_{df} contains a token, such that $|M(i_{df})| > 0$. Now, there must be a binding based on the tokens in the variable places $p1$, $p2$ and $p3$ so that one of the transitions in T_{dt} can fire. Otherwise, the net is stuck (because the table is not conditionally complete).

The second condition of Definition 34 assumes that given that there is a token in i_{df} , indeed one of the transitions in T_{dt} can fire. This leads to the marking M' , which contains a token in $cf1$ in the example. Then, there must be a binding based on the token in the variable place $p3$ so that one of the transitions in T_g can fire. Otherwise, the net is stuck (because the table outputs are not conditionally covered).

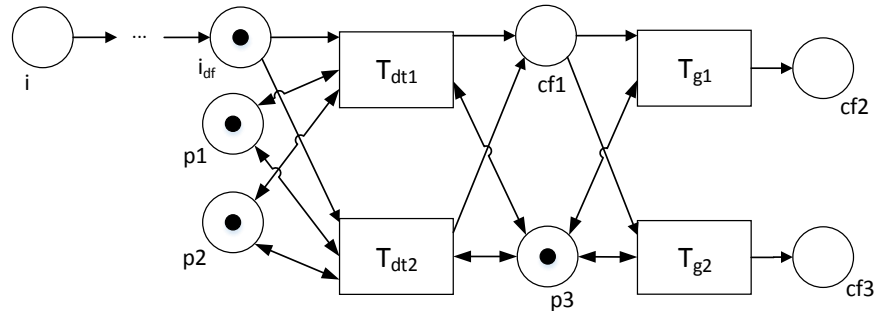


Figure 7.6: Sketch of a CPN decision fragment to visualize Definitions 34 and 35

7.3 STATE-BASED DEAD BRANCH ABSENCE

State-based dead branch absence is based on the same line of argumentation as state-based decision deadlock freedom. Also in this case, state information is utilized to refine the definition and checking of decision-aware soundness. As an example of where the state-based and stateless variants of dead branch absence differ, consider a slightly modified version of the process model in Figure 7.3, namely the one in Figure 7.7.

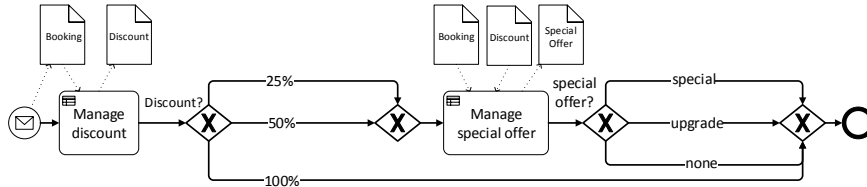


Figure 7.7: Adapted version of the process model in Figure 7.3

The second decision fragment of this model now contains an additional branch succeeding the split gateway, associated with the condition *none*. From a stateless perspective all branches are reachable, because for every branch condition, there is a matching decision table output. From a state-based point of view, however, the output *none* cannot be produced in the context of this process model, such that the corresponding branch cannot be reached—it is *conditionally dead*. This example demonstrates that considering state information for decision soundness checking can render decision fragments either sound or unsound, depending on the criterion (DBA or DDF). This will be discussed further in the next chapter.

State-based dead branch absence (SB-DBA) is then defined as follows.

Definition 35 (State-based dead branch absence). Let CPN_{df} be a colored Petri net fragment representing the decision fragment df , which is embedded into a colored workflow net CPN . Let i_{df} be the initial place of CPN_{df} and M_I be the initial marking of CPN . Further, let M be a marking and b a binding. The decision fragment df satisfies the *state-based dead branch absence* criterion if and only if:

$$(i) \forall t \in T_g, \exists M, M', M'', t' \in T_{dt}, b, b' : M_I \xrightarrow{*} M \wedge |M(i_{df})| > 0 \wedge M \xrightarrow{(t', b')} M' \wedge M' \xrightarrow{(t, b)} M''.$$

◇

The only condition of SB-DBA says that, for all gateway transitions of CPN_{df} , there must be a marking that can be reached after firing a decision table transition, so that the gateway transition can fire.

The example in Figure 7.6 shows a marking M in which i_{df} contains a token. Firing one of the transitions in T_{dt} will lead to a marking M' containing a token in cf_1 . From this marking, firing one of the

transitions in T_g will lead to a marking M'' with a token in cf_2 or cf_3 . If a transition in T_g cannot fire under at least one binding b from M' , the process model contains a conditionally dead branch.

In the last two chapters a stateless and a state-based notion of soundness for decision-aware process models were introduced. While Chapter 6 discussed criteria of decision soundness that consider decision fragments in isolation and are stateless, Chapter 7 explained how state information can be utilized to achieve more practical results. This chapter investigates the relationship between these two approaches to decision soundness and also combinations and variants of the two.

Section 8.1 explores the relationship between the stateless and state-based soundness criteria and Section 8.2 defines various notions of decision soundness motivated by the various traditional notions of soundness introduced in Section 2.1.3.2. Finally, Section 8.3 contains a short summary and discussion. The work presented in this chapter has partly been published in [77].

8.1 RELATIONSHIP BETWEEN THE STATELESS AND STATE-BASED DECISION SOUNDNESS CRITERIA

This section will show that the stateless decision deadlock freedom criterion (DDF) is more restrictive than its state-based variant (SB-DDF), i. e., it is violated in every situation in which SB-DDF is violated *and* others (Section 8.1.1). In contrast, stateless dead branch absence (DBA) is more permissive than state-based dead branch absence (Section 8.1.2).

8.1.1 $DDF \implies SB-DDF$

The decision deadlock freedom criterion has two requirements: first, the corresponding decision model must only contain complete decision tables, and second, the decision fragment branches must cover all of the decision model's outputs (cf. Definition 31). The first condition, table completeness, requires that for every input combination $iv \in IV$ there is a matching rule in the table (cf. Definition 30). By default, the set $IV = \prod_{j=1}^n \text{dom}(i_j)$ is made up of the Cartesian product of the domains of the input variables.

In terms of a state space analysis of the corresponding colored workflow net as described in Chapter 7, this means that there are theoretically as many different markings containing a token in the initial place of the decision fragment CPN as there are elements in IV , and not more, but potentially less. This is because some values of

IV may not occur in the context of a particular process model, and therefore, there are fewer markings that need to be “handled” according to Definition 34 (SB-DDF), than according to Definition 31 (DDF). Therefore, a table may not be complete (cf. Definition 30), but still be conditionally complete (cf. Definition 34), but not the other way round.

From the fact that there are potentially fewer markings before the decision is made, it follows that there are potentially fewer markings after the decision was made. This is analogous to saying that the set of possible outputs of the decision table can only be decreased because fewer rules will match and produce an output. Thus, for state-based output coverage the branch conditions of the decision table need to cover the same or even less outputs than for (non-state-based) output coverage.

Hence, DDF is more restrictive than SB-DDF, such that DDF implies SB-DDF, meaning that every decision fragment that is DDF is also SB-DDF, but not vice versa. An example of this situation was given in Figure 7.3 where the second decision fragment is SB-DDF and not DDF.

8.1.2 $SB\text{-}DBA \implies DBA$

The implication between DBA and SB-DBA is the opposite way, which is not surprising since dead branch absence relates the outputs of the decision model and the branch conditions of the decision fragment exactly opposite to decision deadlock freedom (cf. Definition 32). Consequently, all of the decision fragment’s branch conditions may be reachable through the outputs that can be *theoretically* produced by the decision model—meaning that the decision fragment is stateless dead branch free. However, through state space analysis it may become apparent that only a subset of the rules can be matched in the context of a particular process model, such that only a subset of the decision model’s outputs can be produced. Thus, there are potentially fewer reachable markings after the decision, so that it may *not* be the case that for all branch conditions there is a marking such that this condition is fulfilled (as required by Definition 35). The same decision fragment that is dead branch free may therefore not be state-based dead branch free.

Hence, SB-DBA is more restrictive than DBA, such that SB-DBA implies DBA, meaning that every decision fragment that is SB-DBA is also DBA, but not vice versa. An example of this situation was given in Figure 7.7 where the second decision fragment is DBA and not SB-DBA.

8.2 VARIOUS NOTIONS OF DECISION SOUNDNESS

The previous section suggests that there exist different levels of decision soundness depending on whether or not state-based versions of the criteria are applied. Additionally, it may be possible to weaken the state-based criteria to allow for more models to be sound in certain situations. The idea behind such considerations comes from the taxonomy of traditional soundness notions described in [44] and shown in Figure 8.1a

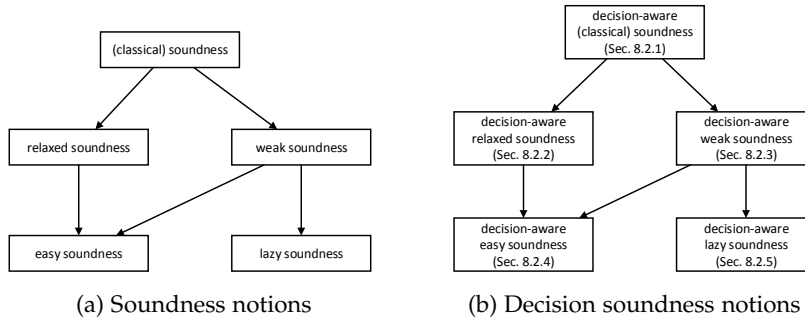


Figure 8.1: Various notions of (decision) soundness

From this taxonomy we derived a corresponding taxonomy for decision soundness shown in Figure 8.1b. Comparing the two figures shows that the relationships between the different notions are preserved. Each derived notion will be defined and illustrated in the following sections.

8.2.1 *Decision Soundness*

This soundness criterion corresponds to the root node in Figure 8.1b. As the root node it must be the most restrictive of all decision soundness notions, implying all of the other notions. Therefore, any process that satisfies decision soundness will, for example, also satisfy the notion of relaxed decision soundness. From the preceding section discussing the relationship between the stateless and state-based variants of the decision soundness criteria, it follows that the most restrictive decision soundness notion must be formulated as follows:

Definition 36 (Decision-aware soundness). A decision-aware process model is decision-aware sound if and only if:

- (i) it is (classically) sound,
- (ii) all of its decision fragments are decision deadlock free, and
- (iii) all of its decision fragments are state-based dead branch free.

◇

Consider the simplified process model in Figure 8.2, with the *Manage special offer* decision shown in Figure 8.3.¹ The process is not

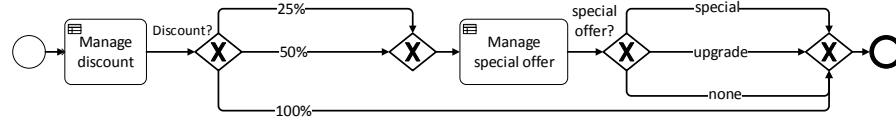


Figure 8.2: Non-decision sound variant of the process model in Figure 7.3

F	BahnCardType {25, 50, 100}	Discount {25%, 50%, 100%}	Special offer {none, upgrade, special}
1	25	25%	special
2	50	-	upgrade
3	-	100%	none

Figure 8.3: Decision table called by *Manage special offer*

decision-aware sound for the following reason: although the second decision fragment seems to perfectly match the associated decision table, state space analysis shows that the output *none* cannot be produced in the context of that process model, so that the branch with that condition is state-based unreachable, violating the third condition of the above definition.

8.2.2 Relaxed Decision Soundness

Relaxed soundness as specified in Definition 14 allows deadlocks to occur during process execution. However, every transition (or activity) must be able to participate in at least one sound firing sequence. This means that “relaxed soundness assumes a responsible user or environment” [44, p. 344], such that the paths leading to deadlocks are not taken during runtime. Conversely, in a decision-aware setting, the decisions which paths to take are explicitly modeled in DMN, and thus the responsibility is now transferred to design time.

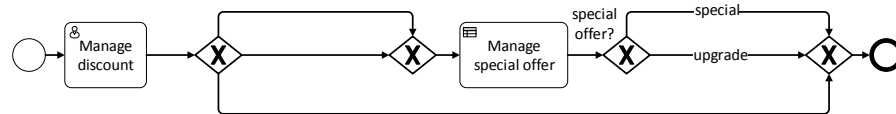


Figure 8.4: Relaxed decision sound variant of the process model in Figure 7.3

As an example, consider the process model in Figure 8.4. In this process the first decision is taken by a user while the second decision is taken by the decision table in Figure 8.3. Therefore, due to the decision-agnostic setting of the first split gateway—where a user takes the decision—a user could choose one of the upper two branches of the split gateway, even if *Discount* = 100%. This in turn

¹ In the remainder of this chapter the data objects are omitted from the process models.

makes it possible to match the third rule of the table in the second decision fragment, leading to a user-induced deadlock.

In contrast, if also the first decision is explicitly taken by a decision model (such as in Figure 7.3), state space analysis can reveal that such a deadlock will never occur at runtime, as discussed above. Therefore, that process model is *by design* decision-aware relaxed sound, because it satisfies the following definition, which is less restrictive regarding decision deadlock freedom compared to Definition 36.

Definition 37 (Decision-aware relaxed soundness). A decision-aware process model is decision-aware relaxed sound if and only if:

- (i) it is relaxed sound,
- (ii) all of its decision fragments are state-based decision deadlock free, and
- (iii) all of its decision fragments are state-based dead branch free.

◇

8.2.3 Weak Decision Soundness

The requirement of weak soundness (cf. Definition 15) is that every process instance can terminate properly. Thereby, deadlocks and remaining tokens are disallowed. What separates weak soundness from classical soundness is the fact that dead activities are permitted. Consequently, decision-aware weak soundness is defined as follows:

Definition 38 (Decision-aware weak soundness). A decision-aware process model is decision-aware weak sound if and only if:

- (i) it is weak sound,
- (ii) all of its decision fragments are state-based decision deadlock free, and
- (iii) for all of its decision fragments there exists a path that is state-based dead branch free.

◇

In this definition, the second condition requires state-based decision deadlock freedom to prevent deadlocks. The third condition requires state-based dead branch absence, but only for at least one “path”. This simply means that the universal quantification of Definition 35 over the set of transitions $t \in T_g$ is turned into an existential one, i. e., formally, the only difference is that the “ \forall ” in Definition 35 is replaced by an “ \exists ”. This allows the decision fragments to contain dead branches, but requires at least one branch to be reachable for

each decision fragment, so as to enable the proper termination of process instances.

Note that universal quantification is more restrictive than existential quantification. Therefore, all conditions of weak decision soundness are more permissive than those of decision soundness, such that the implication between the two in Figure 8.1b is satisfied.

The process model in Figure 8.2 is decision-aware weak sound. The branch of the second decision fragment with the condition *none* is state-based dead, because the output *none* cannot be produced. But the other branches are still reachable, such that the existential requirement of the third condition is fulfilled. However, the model is not decision-aware relaxed sound, because of that one dead branch.

8.2.4 Easy Decision Soundness

A workflow net is considered easy sound if a state of proper completion can be reached at least once. This means that both deadlocks as well as dead activities are allowed, such that easy soundness is implied by relaxed and weak soundness at the same time. In terms of the decision-aware variant of this criterion, guaranteeing proper completion at least once requires at least one path through the model that is both state-based decision deadlock free and state-based dead branch free.

Similar to the existential quantification of SB-DBA, the existential quantification (“one path”) of SB-DDF means that the universal quantifiers in Definition 34 are replaced by existential quantifiers. Therefore, there must only exist at least one value such that some decision table rule can be executed and there must only exist at least one output that is covered by the branch conditions.

Hence, the following definition:

Definition 39 (Decision-aware easy soundness). A decision-aware process model is decision-aware easy sound if and only if:

- (i) it is easy sound, and
- (ii) for all of its decision fragments there exists a path that is state-based decision deadlock free and state-based dead branch free.

◇

Notice that, strictly speaking, the second conjunct of the second condition requiring state-based dead branch absence is not necessary, since the existence of a path that is state-based decision deadlock free implies that this path is also state-based dead branch free.

Consider the process in Figure 8.5, where the decision tasks reference the tables in Figure 8.6. Note that the second rule of the first decision table does not consider the case when *BahnCardType* = 50

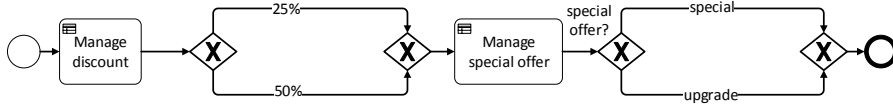


Figure 8.5: Easy decision sound variant of the process model in Figure 7.3

U	BahnCardType {25, 50, 100}	Discount {25%, 50%, 100%}
1	25	25%
2	25	50%
3	100	100%

(a) *Manage discount* decision table

U	BahnCardType {25, 50, 100}	Special offer {none, upgrade, special}
1	25	special
2	50	upgrade
3	100	none

(b) *Manage special offer* decision table

Figure 8.6: Decision tables referenced by the decision tasks of the process model in Figure 8.5

anymore. This means it is incomplete and therefore violates (state-based) decision deadlock freedom. Furthermore, given that the *upgrade* branch of the second decision fragment can only be taken given that *BahnCardType* is 50, this branch is state-based dead.

It follows that this process is neither decision-aware relaxed nor weak sound. Still, there is a path through the process and the decision tables that leads to proper completion, namely the path that is taken in case $BahnCardType = 25$. Therefore, the process is decision-aware easy sound.

8.2.5 Lazy Decision Soundness

Lazy soundness disallows deadlocks and allows dead transitions. Additionally, it allows remaining tokens. This means that when a token reaches the final place, there can be tokens left in the net and there can be transitions that are enabled or fire. However, the final place should only be reached once. Lazy decision soundness can be defined as follows:

Definition 40 (Decision-aware lazy soundness). A decision-aware process model is decision-aware lazy sound if and only if:

- (i) every process instance reaches the end event exactly once, and
- (ii) for all of its decision fragments there exists a path that is state-based decision deadlock free and state-based dead branch free.

◇

Regarding the decision criteria, this definition is the same as that for easy decision soundness (cf. Definition 38). The difference is that it (clearly) does not require easy soundness, but more importantly it also does not require *lazy* soundness. Still, it requires that every instance reaches the end event exactly once.

This subtle variation is due to the fact that the use of decision models can render process models lazy decision sound, that are not considered lazy sound in the sense of Definition 16. As an example, study the process model in Figure 8.7, which is not lazy sound: The parallel gateway spawns two concurrent paths which are later joined by an exclusive gateway right before the end event, leading to two tokens in the final place.

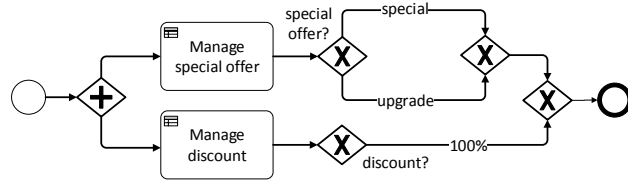


Figure 8.7: Lazy decision sound variant of the process model in Figure 7.3

Now assume that the two decision fragments reference the decision tables in Figure 8.6a and Figure 8.6b, respectively. With this configuration, in every possible instance, exactly one of the concurrent branches contains a remaining token: if *BahnCardType* = 100 a token remains in the upper branch; and if the type is 50 or 25 the lower branch contains a remaining token. Consequently, each process instance eventually reaches the end event exactly once, such that the process model is decision-aware lazy sound.

8.3 SUMMARY AND DISCUSSION

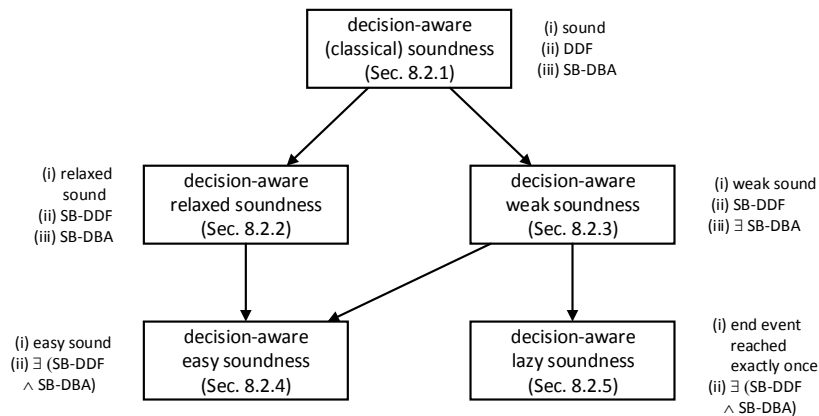


Figure 8.8: Various notions of decision soundness annotated with their respective conditions as defined in Section 8.2

The various notions of decision soundness described above are summarized in Figure 8.8. For each notion the figure shows the conditions that must be fulfilled in order to satisfy that notion. The numbering corresponds to the numbering of the conditions of the corresponding definition in Section 8.2.

This figure also demonstrates that the implications between the different notions (indicated by arrows) are satisfied by the conditions. For example, there is an arrow between relaxed decision soundness and easy decision soundness, claiming an implication between the two. Indeed, relaxed soundness (of workflow nets) implies easy soundness (cf. Figure 8.1a), and if SB-DDF and SB-DBA are fulfilled, then there certainly exists at least one path that is SB-DDF and SB-DBA. Therefore, the implication is true, and similar arguments can be made for the other implications in Figure 8.8.

Regarding the actual verification of the decision soundness criteria, it is important to note that this will be done in two steps. For example, to check relaxed decision soundness, first, the formalization of the process model as a workflow net will be checked for relaxed soundness. Afterwards, the formalization of the process model as a *colored* workflow net will be checked for relaxed decision soundness.

This demonstrates that the various notions of *decision soundness* differ from the various notions of *soundness* that could be defined for colored workflow nets. Definition 28 already gives a definition of soundness for a colored workflow net, and similarly one might define a notion of relaxed soundness of a colored workflow net. Without giving a full definition here, we claim that that definition will (at least) replace criteria (i) – (iii) of colored workflow net soundness by the following criterion:

$$(i) \quad \forall t \in T, \exists M_f \in M_F, M, M', b : M_I \xrightarrow{*} M \xrightarrow{(t,b)} M' \xrightarrow{*} M_f.$$

This criterion allows state-based decision deadlock freedom to be violated. It only requires every transition to be part of a sound firing sequence of the colored workflow net, i. e., there can be markings anywhere in the net that are deadlocks. Relaxed decision soundness also allows deadlocks (due to its first condition), but not in the decision fragment (due to its second condition). Therefore, a colored workflow net that is relaxed sound may not be relaxed decision sound, and similar arguments can be made for the other soundness notions.

Part IV

EVALUATION AND CONCLUSION

EVALUATION

This chapter is concerned with the evaluation of the concepts and ideas described in this thesis. First, the prototypical implementation of the algorithms necessary for checking the (stateless) soundness of decision-aware process models are outlined in Section 9.1, including runtime considerations. Next, an empirical evaluation of the notion of decision soundness is presented with models from participants of an online course on process and decision modeling as well as from a BPM project of a large insurance company in Section 9.2. Finally, Section 9.3 demonstrates how the explicit consideration of decisions in process models can improve also another area of analysis, namely compliance checking. The work presented in this chapter has partly been published in [78–80, 95, 96, 101].

9.1 PROTOTYPICAL IMPLEMENTATION

In order to enable the verification of (stateless) soundness of decision-aware process models, we implemented corresponding algorithms in *dmn-js*¹, a DMN decision table editor developed by Camunda. Those algorithms were already described in [79] and [95], and their combined use integrated into the *Camunda Modeler*² will be presented in the following.

9.1.1 Extended Camunda Tool

The entire functionality of the soundness checking procedure was implemented in the *dmn-js* component of the Camunda Modeler, a standalone application for the integrated modeling of BPMN and DMN diagrams. The Camunda Modeler allows to design process and decision models whose serialization in XML format is conformant to the respective standards. Therefore, decision soundness can be checked for models that were created with the Camunda tool itself or, alternatively, for models that were created with a different standard conformant application and that are then imported to Camunda. Our extended, ready to execute Camunda Modeler and corresponding screencasts are available for download.³

Figure 9.1 shows a screenshot of the extended tool after the soundness check was executed for the decision-aware process model at the

¹ <https://github.com/bpmn-io/dmn-js>

² <https://github.com/camunda/camunda-modeler>

³ <https://bpt.hpi.uni-potsdam.de/Public/BpmnDmnSoundness>

bottom of that figure. This simplistic process determines the credit rating of an applicant and, depending on the result, accepts or rejects the applicant or invites them for an interview. The credit rating is determined by the decision table displayed at the top of the figure and the analysis results are shown in the middle.

The table in this example is actually the same as the one in Figure 4.1 in Section 4. The analysis results in the screenshot show that the output set of the table is computed to be able to check decision deadlock freedom and dead branch absence. Note that in this prototypical implementation the stateless variants of the criteria defined in Section 6.3 are checked. Therefore, multiple violations are detected, five of which are due to a DDF violation and the other one being a DBA violation.

The results also show the reasons for the violations. For example, the table can produce the output $[A, B]$, but there is no branch condition in the process model that would take care of this output. The first condition requires the output to be exactly equal to A , the second expects the single value E or a list that contains the value E , and the third requires the value A *and* a list that does not contain E or a single value that is not equal to E .⁴ Similar arguments apply to the other instances of DDF violations.

The last violation concerns the lowest branch of the process model, which is unreachable because it requires the decision output to be both a single value (equal to A) and at the same time a list that does not contain E . Hence, the dead branch absence criterion is not satisfied. Luckily, the solution to all of those problems is simple. The designer of the process model merely forgot to negate the first part of the condition and confused the logical *or* in the last part with a logical *and*. The correct branch condition would thus be:

$$\text{CreditRating} \neq A \text{ and } (\text{CreditRating} \text{ !contains } E \text{ and } \text{CreditRating} \neq E)$$

which simply matches everything that does not match the first two conditions. Updating the process model accordingly would then yield the result that it is indeed (stateless) decision sound.

Notice that the decision table output set that is displayed in the first row of the analysis results table is computed by the procedure described in Section 4.2, but the table is not uniqueified (i. e., transformed into a unique table) afterwards. However, this can also be achieved by pressing the *uniqueify table* button on the top right of the figure. The result is shown in the screenshot in Figure 9.2.

9.1.2 Runtime Analysis

Parts of our implementation rely on functionality provided by another extension of *dmn-js* for the verification of DMN decision ta-

⁴ The violations are obviously due to that last branch condition.

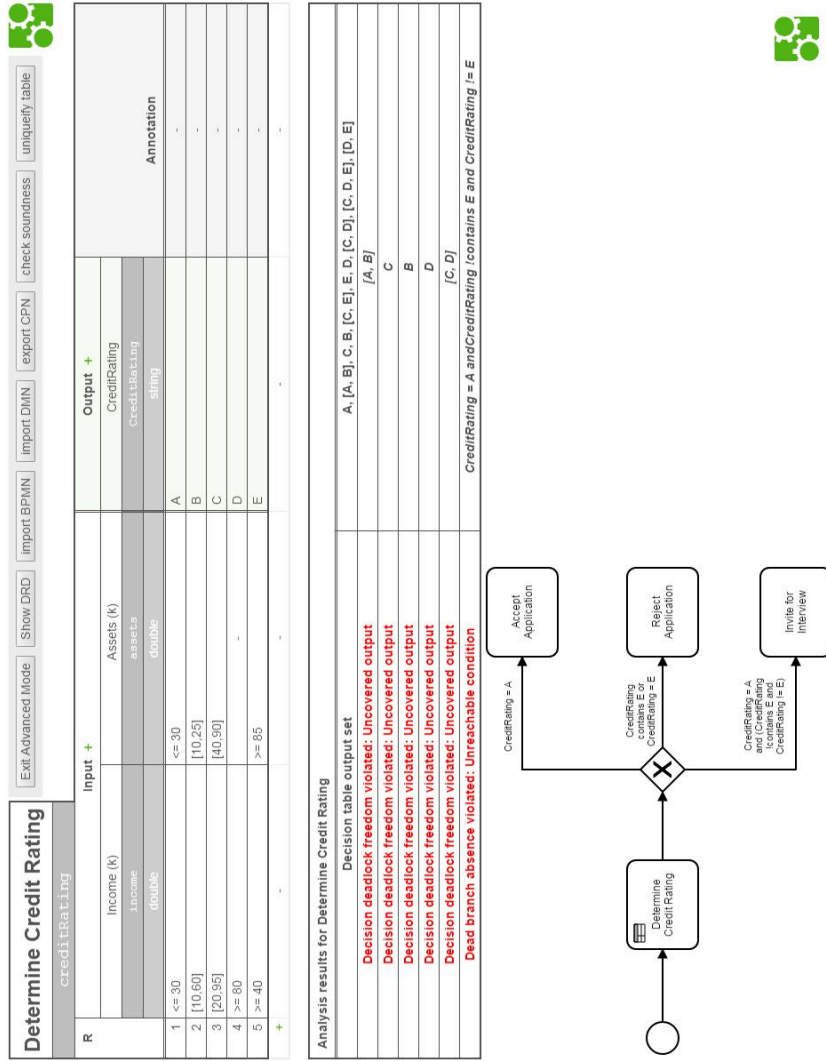


Figure 9.1: View of the tool after checking soundness of the displayed decision-aware process model

Determine Credit Rating		Input +		Output +		Annotation
CreditRating		Income (K)	Assets (K)	CreditRating	CreditRating	
		Income double	assets double	CreditRating string	CreditRating	
1	< 10		<= 30		A	-
2	[10, 30]		< 10		A	-
3	[10, 30]		[10, 25]		A,B	-
4	[10, 30]		(25, 30]		A	-
5	[20, 40]		[40, 90]		C	-
6	(30, 60]		[10, 25]		B	-
7	[40, 80]		[40, 85]		C	-
8	[40, 80]		[85, 90]		C,E	-
9	[40, 80]		> 90		E	-
10	[80, 95]		< 40		D	-
11	[80, 95]		[40, 85]		C,D	-
12	[80, 95]		[85, 90]		C,D,E	-
13	[80, 95]		> 90		D,E	-
14	> 95		< 85		D	-
15	> 95		>= 85		D,E	-

Figure 9.2: View of the tool after uniquelyifying the table shown in Figure 9.1

bles [73]. That extension includes the detection of missing and overlapping rules, and the optimization of the table by merging rules. In [97] the algorithms are reported to be of quadratic complexity. Checking decision deadlock freedom and dead branch absence does not result in a higher complexity class. This is because both checks essentially consist of two nested for-loops that compare the decision table outputs with the process branch conditions, which is also of quadratic complexity.

The most time-consuming part of the decision soundness check is actually the computation of the decision table output set—if the table contains overlapping rules. To assess the runtime of that computation, 1000 different artificial decision tables with overlapping rules were generated and uniqueified. Those 1000 tables are made up of ten sets of 100 decision tables, where each set contains tables with all combinations of $\{5, 10, 15, \dots, 50\}$ rows and $\{3, 6, 9, \dots, 30\}$ input columns columns. All the input columns of the table have an integer domain, while the single output column is of type string.

The rules are randomly generated by generating a random conditions for each input. Each condition is made up of one of the operators in $\{<, >, \leq, \geq, =\}$ and an integer, both chosen at random. Moreover, each table has a rule-order multi-hit policy. Hence, the generated tables have a random number of overlapping rules and can therefore be processed by the uniqueification algorithm to compute their output sets.

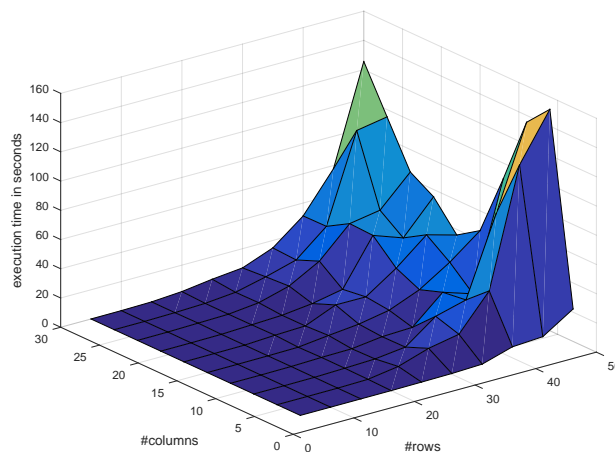


Figure 9.3: Averaged execution times in seconds for a set of 1000 multi-hit tables with up to 50 rows and 30 columns

The algorithm was applied to the tables in the ten sets and the execution times were averaged over the ten sets. The measurements are displayed as a three-dimensional surface plot in Figure 9.3, where the x-axis shows the number of rules of the table, the y-axis the number of columns, and the z-axis the averaged execution times.

The plot shows that the execution times are more dependent on the number of rows than the number of columns. This is because the more rules there are, the more intervals need to be checked for matching rules. However, the maximum average execution time was actually reached for the 50×6 -tables with 151 seconds; and also other tables with similar numbers of rules and columns lead to comparably high execution times.

The reason for that is the high number of overlapping rules of those tables. More specifically, tables with a high number of rules but a low number of columns have much more rules that overlap across all dimensions. And the more rules overlap in all dimensions, the more intervals need to be checked per dimension (cf. recursive call in Algorithm 1).

The average total number of overlaps of the rules of the tables in our data set is shown in Figure 9.4. Here, the maximum number of overlaps is 1138 for the 50×6 -tables, which coincides with the maximum in Figure 9.3.

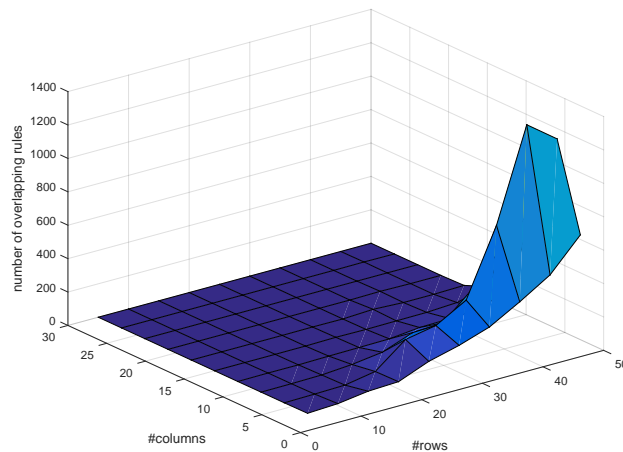


Figure 9.4: Total number of overlaps found in the tables

Note that the number of overlaps for the tables with a high number of columns is nearly always zero. This is because the higher the number of columns, the harder it is for multiple rules to overlap in *all* of those dimensions. This shows that our uniqueification approach is especially suited (or effective) for tables with a high number of rules and a lower number of columns.

9.2 EMPIRICAL EVALUATIONS

This section first describes in Section 9.2.1 the results of an evaluation of the decision soundness criteria of Section 6.3 on models obtained from participants of an online course. Afterwards, Section 9.2.2 presents results and statistics of the analysis of a large real-world BPM project. Both empirical evaluations were conducted to find out


whether process and decision designers indeed make modeling mistakes that would lead to a violation of the decision soundness criteria, thereby ascertaining their practical relevance.

9.2.1 Academic Models

The collection of “academic” decision-aware process models came from participants of a massive open online course (MOOC) on process and decision modeling with BPMN and DMN offered by *openHPI* [56] in Spring 2016.⁵ The MOOC lasted six weeks and had around 5500 participants in total, the majority of which were between 30 and 50 years old working as consultants with an IT background, while the second largest group came from academia. For the successful completion of the course, the participants had to solve weekly exercises involving both theoretical questions and modeling tasks.

Modeling Task 1

Edit item

 This quiz is a self-test. You can repeat it as often as you like.

Question 1

1.0 Pts

An energy provider has a billing process to send out monthly bills to its customers. For this reason, the company needs to decide which type of billing plan the bill should be based on.

There are bills containing a minimum fee, bills according to plan A, and bills according to plan B. If the customer has a fixed rate contract, a minimum monthly charge is assessed for consumption of less than 1000 kwh. Otherwise, the customer receives a plan A bill. If the account is billed using a variable rate method, a plan A bill will apply to consumption below 2500 kwh. At the same time, any additional consumption will be billed separately according to plan B.

After the decision was made, the energy provider sends out different letters for each bill. Then, the billing process ends.

Did you upload your models as an .sgx or .zip archive [here](#)?

no

yes

[< Previous](#) [▶ Send my answers](#) [Next >](#)

Figure 9.5: Modeling exercise given to openHPI course participants

To evaluate the decision soundness criteria, in the last week of the course the participants were asked to voluntarily submit their solution to a modeling exercise requiring them to model a BPMN process associated with a DMN decision model. For that purpose, they were given a textual description of a simplistic billing process, in which a decision is made about the particular type of bill that applies to a cus-

⁵ <https://open.hpi.de/courses/bpm2016>

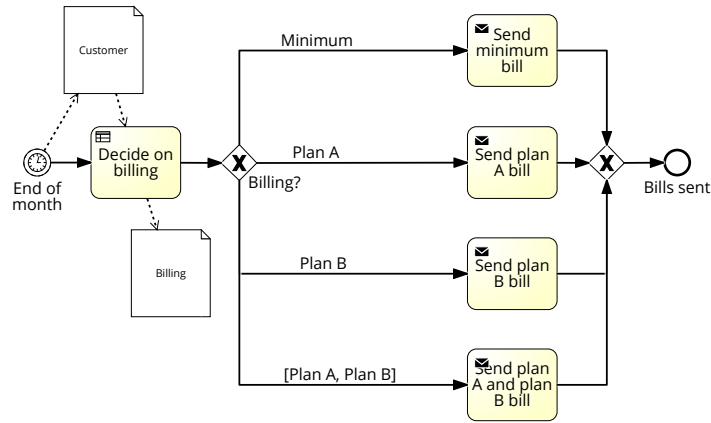


Figure 9.6: Model for the billing process described in Figure 9.5

R	Inputs			Outputs
	Fixed rate	Consumption		Billing
	Boolean	Number		{Minimum, Plan A, Plan B}
1	=	true	< 1000	Minimum
2	=	true	≥ 1000	Plan A
3	=	false	-	Plan A
4	=	false	≥ 2500	Plan B

Figure 9.7: Decision table for the billing decision described in Figure 9.5

tomer. A screenshot of the openHPI system displaying the exercise is shown in Figure 9.5.

Figure 9.6 and Figure 9.7 show the sample solution for the modeling task in Figure 9.5. The process is started at the end of the month and the type of the bill is decided upon based on the customer data. As described in the text, the decision table takes into account whether or not the customer has a fixed rate contract and their consumption. Note that rules three and four are overlapping for consumption values of more than 2500 kWh. Therefore, the output set of this decision table is *Minimum, Plan A, Plan B, [Plan A, Plan B]*. The last output is a list output and is processed by the decision fragment accordingly. In this way, two bills are sent for such cases, as asked in the text.

From the course participants, in total, we received 157 unique submissions, 70 of which were evaluable with respect to decision-aware soundness. The two most common reasons for the other 87 submissions to be unsuitable for our evaluation were that either the participants only submitted a DMN model but no associated BPMN model (52 submissions), or that the associated BPMN model did not contain a decision fragment as defined in Definition 20 (24 submissions). Rather, the process was designed like in Figure 2.22.

Table 9.1 summarizes the results of our analysis. While the sample solution in Figure 9.7 uses a multi-hit table, more submissions favored

Table 9.1: Number and percentage of sound and unsound decision-aware process models designed by the MOOC course participants

	Sound	Not Sound	Total
Single-hit	12 (29%)	29 (71%)	41 (100%)
Multi-hit	15 (52%)	14 (48%)	29 (100%)
Total	27 (39%)	43 (61%)	70 (100%)

using single-hit tables, which is also possible. Looking at the ratio of unsound processes, in both cases mistakes are made: 71% of the single-hit and 48% of the multi-hit tables were found to be unsound. Thus, in total, 61% of the analyzed decision fragments violate at least one of the decision soundness criteria of Section 6.3.

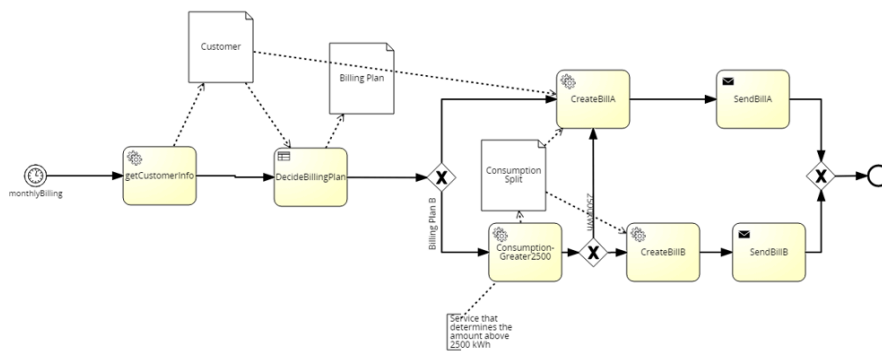


Figure 9.8: Submission by a course participant for the modeling task described in Figure 9.5

U	Inputs		Outputs	
	Customer.billingplan	Customer.consumption	Billing Plan	
	{fixed, variable}	Number	{minimum, A, B}	
1	= fixed	< 1000	minimum	
2	= fixed	≥ 1000	A	
3	= variable	< 2500	A	
4	= variable	≥ 2500	B	

Figure 9.9: Decision table associated with the decision fragment in Figure 9.8

Figure 9.8 and Figure 9.9 show a “typical” solution by a course participant. Since all the rules of the table are exclusive, the output set is *minimum, A, B*. But only for the *B* output there is a corresponding branch condition (*Billing Plan B*). The other outputs are not explicitly considered here (i. e., output coverage is violated). One may assume that the upper branch of the first split gateway simply matches anything that is not equal to *Billing Plan B*. But this does not seem feasible, because that branch only leads to an activity dealing with a *Plan A* bill, and not with a *Minimum* bill.

In conclusion, most participants understood how to design decision-aware business process models, i. e., they modeled decision fragments that interact with decision models. However, they lacked the precision required for a sound integration of the two types of models. This is especially important when such models are supposed to be executed automatically in an environment where there are no domain experts that could guess how the model should be interpreted correctly.

9.2.2 *Industry Use Case*

Since the empirical evaluation of the previous section was conducted on models of an artificial use case by modeling novices, another evaluation was carried out on real-life decision-aware process models. Those came from an implemented process management project of a large German insurance company that deals with automatically handling invoices submitted by customers with private health insurance.

The decision soundness evaluation was part of a bigger project investigating the gap between academic research on decision management and decision management in the BPM software Pega by assessing the model quality of an implemented decision-heavy Pega project. Pega is a software platform developed by Pegasystems, intended for the model-driven development of BPM applications.⁶

The scrutinized Pega project is embedded in an application of the insurance company dealing with the reimbursement of health care services filed by their customers. That application's objective is to increase the rate of automation of processing customer requests. In general, the application contains a deterministic and a stochastic component, both of which execute the processes fully automatically or, in some exceptional cases, with the help of a case worker. The deterministic component is implemented in Pega and includes, for example, checking the filed requests for correctness according to formal and legal requirements, and calculating the reimbursement amount according to the tariff of the customer.

The Pega project contains 86 decision-aware process models, 26 (30%) of which were found to be not decision sound because they violate the output coverage criterion. An example of such a process model is shown in Figure 9.10, which is associated to the decision table in Figure 9.11 via its gateway (*Determine Block Number*).

The process model is designed to react to two outcomes of the table, *Block 2* and *Block 3*. However, the decision table can also produce another output, namely *Error*, which is not covered by the process model. Hence, the decision deadlock freedom criterion is violated and the process is not decision sound.

Yet, the state-based version of the decision deadlock freedom criterion is actually *not* violated by this process. This was discovered

⁶ <https://www.pega.com/>

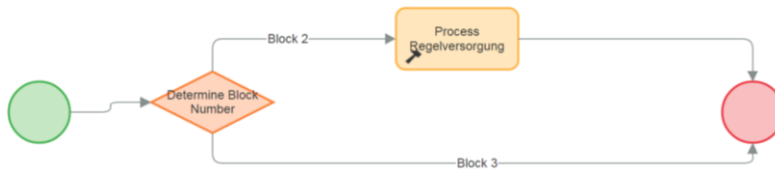


Figure 9.10: Example of decision-aware process model found in the analyzed Pega project

	Conditions	Actions
	◦ BlockNumber	Return
◦ if	2	→ 2
◦ else if	3	→ 3
otherwise		→ Error

Figure 9.11: Decision table associated with the gateway in Figure 9.10

after a consultation with one of the process owners who explained that the application in which the Pega project is embedded already makes sure that the variable *BlockNumber* will only contain either the value 2 or 3. Hence, the process is *state-based* decision deadlock free and, therefore, satisfies all of the notions of decision soundness in Figure 8.1b, except for the top one.

Apart from the 86 decision-aware process models, the Pega project also contains decision tables that are not associated with any processes. While they cannot be evaluated with regard to decision soundness, they can still be analyzed regarding completeness. When doing so it makes a substantial difference how completeness is defined. In general, a table is complete if for every possible combination of input values at least one rule matches. The question then is what the possible input values are.

According to DMN, the possible input values are the values of the variable’s domain, which can be infinitely many, in case of an integer domain, for instance. Pega, on the other hand, assumes that only those values that are actually used in the table are the values that the variable can assume.

For example, the table in Figure 9.12 is complete according to Pega, but not according to DMN, because the variable *RECHPRUEERG* is of

	Conditions		Actions
	◦ RECHPRUEERG	◦ DE1_STRGNRABWEICH	Return
◦ if	5400	!=0	→ true
◦ else if	5400	=0	→ false
◦ else if	5500	!=0	→ true
◦ else if	5500	=0	→ false
otherwise			→ Error

Figure 9.12: Example for a Pega decision table taken from the analyzed project

type integer.⁷ Note that this is exactly the difference between stateless and state-based decision deadlock freedom. Applying both completeness notions to the project’s 65 decision tables shows that according to Pega 48 (74%) of the tables are (state-based) complete, but according to DMN only 5 (8%) are (stateless) complete, as visualized in Figure 9.13.

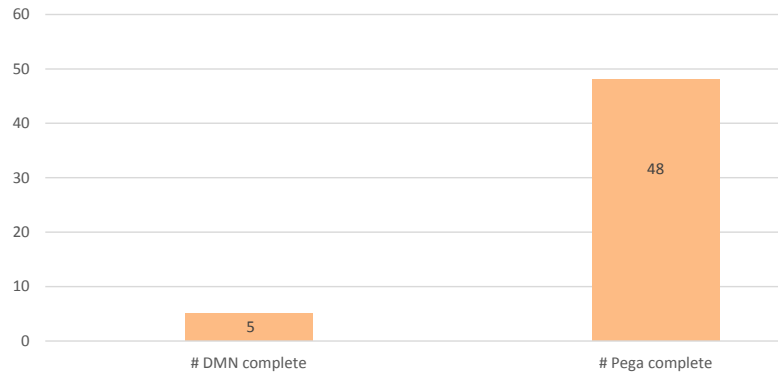


Figure 9.13: Number of decision tables of the Pega project that are complete according to DMN (stateless) and Pega (state-based)

9.3 COMPLIANCE CHECKING OF DECISION-AWARE PROCESSES

Soundness checking is not the only kind of verification that can benefit from a “decision-aware analysis” of the process model. In [101] it was shown that also the verification of process models with regard to compliance rules can take advantage of decision-awareness. The essence of compliance checking is traditionally the verification of rules constraining the control flow of a process model [29]. Such rules may be internal to the organization the process belongs to. But more often the rules stem from laws and regulations that the organization must adhere to. Therefore, compliance checking of process models is the topic of numerous publications (e. g., [27], [29], [33], [38], [92]).

Consider again the running example of a train ticket booking process model, repeated for convenience in Figure 9.14. An internal compliance rule regarding this process may be formulated as follows:

If a 50% discount is applied to the booking, the customer must not be offered a special deal afterwards.

This rule concerns the two activities *Apply 50% discount* and *Offer special deal*. Specifically, it says that if in a particular process instance *Apply 50% discount* is executed, *Offer special deal* must not be executed afterwards in this instance. Such relations between activities can be

⁷ This assumes that the *otherwise* row is not considered in the completeness check, which is the case according to Pega.

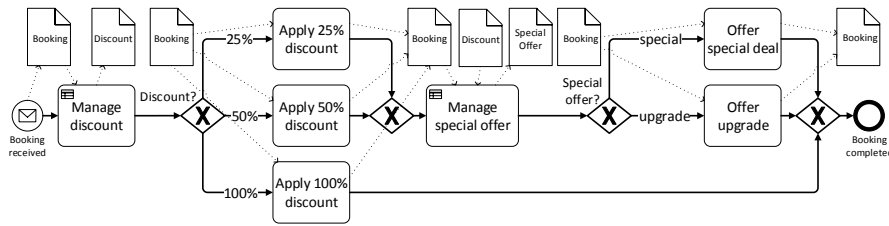


Figure 9.14: The train ticket booking process model, repeated for convenience

formally specified using temporal logic [29]. To explain the impact of decision-awareness, however, it should suffice to work with natural language.

Essentially, to check such a rule one has to analyze the state space of the corresponding process to find out if there are states that violate the rule. For example, one could translate the process model in Figure 9.14 to a workflow net and construct its reachability graph. Then, it will become apparent that there is a path in the graph containing the transition corresponding to *Apply 50% discount* and the transition corresponding to *Offer special deal* afterwards. Therefore, the rule would be violated according to that analysis.

A decision-aware analysis, however, changes that. Translating the process model to a colored workflow net considering the decision logic, and then constructing the reachability graph, will show that there actually is no such path. This is because if a 50% discount is applied, the *BahnCardType* must be 50, according to the first decision table (cf. Figure 7.2a). This in turn means that the output of the second decision table will be *upgrade* (cf. Figure 7.2b), so that the activity *Offer special deal* cannot be executed. Thus, the compliance rule is indeed satisfied.

It turns out that, in general, decision-awareness reduces the number of false negatives and false positives of compliance checking, because it reduces the number of possible traces of a process model—traces that can never occur in reality.

CONCLUSION

The conclusion of this thesis summarizes its results in Section 10.1. Lastly, Section 10.2 discusses limitations of our work, while also providing directions for future work.

10.1 SUMMARY

This thesis introduced and detailed approaches to verify the soundness of a decision-aware business process model, which is a process model that references one or more decision models. Upon being called by the process, the decision model will execute its decision logic and return the result back to the process, which then acts accordingly. This interaction is prone to errors, leading to soundness violations of the process model. Therefore, this thesis defined so called decision soundness criteria along with a method to verify them such that the correctness of a decision-aware process model can be ascertained. In detail, the results of this thesis are the following:

- *The separation of concerns of processes and decisions:* An analysis of almost 1000 real world process models testified that process modeling constructs are regularly misused to implement the logic of decisions that need to be made during process execution. Such implementations are mainly based on three control-flow decision patterns which we elicited and defined.
- *Input-output behavior of DMN decision tables:* Decision tables are the standardized means of defining decision logic in DMN. Due to the different variations of those tables, however, their input-output behavior might not always be clear. Moreover, in terms of their formalization in the context of a process model, it is useful to have a standard format for the tables. Hence, we devised and implemented an algorithm to uniqueify DMN tables, where the uniqueified form serves as their standardized representation. Furthermore, we proved what the maximum number of outputs of a DMN table with a single input column is.
- *Formalization of decision-aware process models and their verification via abstraction:* For a formal verification of decision soundness, a formal representation of a decision-aware process model is required. We proposed a mapping of such models to colored Petri nets, while also acknowledging that soundness verification given a “one-to-one” mapping is undecidable, due to the infinity of the reachability graph. Therefore, an abstraction method

was proposed that divides the infinitely many states of the reachability graph into finitely many partitions. This enables a soundness analysis of the abstract graph to verify the correctness of the original decision-aware process model.

- *Stateless decision soundness*: For a quick decision soundness check based directly on the decision-aware process model, we define structural and basic behavioral criteria. Those criteria consider each decision fragment in isolation and do not take the context of the process model into account which they are embedded in. This may be relevant, however, since the process determines the possible inputs that the decision is called with, and accordingly not all rules of the decision table may be matched. Such situations can particularly arise if the same decision model is used by several slightly different process models, such that the decision model will not perfectly fit with every process model it is associated with.
- *Various notions of decision soundness*: To support the reuse of decision models across multiple process models in the soundness check, we formally defined various notions of decision soundness that also take the context (or states) of the decision fragment into account. Those soundness notions are checked on the abstract colored Petri net mapping of the decision-aware process model. The notions have been inspired by the existing notions of control-flow soundness and have the same relationships.
- *Implementation and empirical evaluation*: We implemented the stateless decision soundness check in the Camunda Modeler, including the table uniqueification algorithm, which is required to compute the possible outputs of the decision table, but can also be executed on a table alone to transform it into its standard format. Moreover, two empirical evaluations were conducted on a set of decision-aware process models designed by participants of an online course on process and decision modeling, and on models from a real world BPM project of a large insurance company. In both cases it was shown that decision soundness violations frequently occur. At the same time, in the real world BPM project it was revealed that taking state information into account renders some of the models indeed decision sound.

10.2 LIMITATIONS AND FUTURE WORK

The work presented in this thesis does not come without limitations that may be subject to future work and will be described in the following.

First of all, the decision soundness considerations of this thesis assume the decision logic to be represented by default as a decision table. This raises the question whether the ideas presented here can also be applied if other forms of decision logic are used. Section 4.3 described how decision tables can be interpreted as functions. Similarly, any other logic that is interpretable as a function may replace decision tables. Regarding the stateless criteria, it even suffices to know what the possible outputs of that function are (i. e., its image), and whether or not it is a partial function (so that its “completeness” can be determined). For the state-based criteria—requiring a mapping to a colored Petri net—the full function definition must be available, such that the decision function is simply mapped to a single CPN transition that implements that function.

Moreover, the mapping of a decision fragment to a colored Petri net assumes that the fragment’s split gateway is of type XOR with non-overlapping conditions. The requirement of an exclusive gateway simplifies the mapping considerably because the inclusive or join is hard to tackle in general. Kunze and Weske [72] demonstrate a mapping in case the split and join gateways are “well structured”, but no general CPN mapping is proposed.

Related to this is the requirement that the conditions of the outgoing edges of the decision fragment’s split gateway are non-overlapping, i. e., at most one condition evaluates to true in a given situation. Regarding the BPMN execution semantics, one could resolve such ambiguities by stating that the edge conditions are evaluated from top to bottom, and the first edge whose condition evaluates to true will be signaled (similar to the first hit policy of a DMN table). Such logic would have to be implemented in the CPN as well.

As already discussed in Section 5.5, the CPN mapping and reachability graph analysis is only possible if the decision logic is implemented in S-FEEL. However, also the stateless soundness check is limited to S-FEEL because the table uniqueification algorithm—which is also used to determine the possible outputs of the table—is limited to S-FEEL. This is because the analysis of DMN tables via their geometric interpretation as hyperrectangles requires S-FEEL tables, since FEEL would also allow the comparison of two variables (e. g., $x < y$) as rule conditions. Such rules cannot be interpreted geometrically, however [97].

In Chapter 9 we described a prototype that is able to check stateless decision soundness. The verification of the state-based soundness notions is not supported, however. Obviously, this requires the decision-aware process model to be translated to an abstract colored Petri net, whose reachability graph can then be generated and analyzed. Tool support for colored Petri net modeling and analysis is given through CPN Tools [21]. Clearly, representing abstract symbols in CPN Tools—which is based on the functional programming language Standard

ML [10]—is a challenge. Representing the color set of an abstract token explicitly is impossible because one would need to define all elements of the powerset of the variable’s original domain. This powerset is infinite, however, if the original domain is infinite (such as the integers). Alternatively, one can represent each element of the powerset as a list of intervals. For example, the element $\{3,7,8,9\} \in 2^{\mathbb{Z}}$ can be represented as $[[3,3],[7,9]]$. Consequently, the color set of the corresponding token would just be a list of such intervals.

Another limitation—one that is not directly related to soundness verification—is the restriction of the proof in Section 4.4 about the maximum number of DMN decision tables to one input column. Certainly, statements about tables with more than one input column would be helpful to get an idea of the maximum number of outputs one has to cover in the process model. Yet, as stated in Section 4.4 for this a complicated analysis of the different ways in which a set of n -dimensional hyperrectangles can overlap is required. Since this does not directly contribute to the main objectives of this thesis, it is not considered here.

BIBLIOGRAPHY

- [1] Carl Adam Petri. "Kommunikation mit Automaten." ger. PhD thesis. Universität Hamburg, 1962.
- [2] D. L. Parnas. "On the Criteria to Be Used in Decomposing Systems into Modules." In: *Commun. ACM* 15.12 (Dec. 1972), pp. 1053–1058. ISSN: 0001-0782.
- [3] Herbert Stachowiak. *Allgemeine Modelltheorie*. Wien, New York: Springer, 1973.
- [4] Peter Pin-Shan Chen. "The Entity-relationship Model—Toward a Unified View of Data." In: *ACM Trans. Database Syst.* 1.1 (Mar. 1976), pp. 9–36. ISSN: 0362-5915.
- [5] J. L. Bentley and T. A. Ottmann. "Algorithms for Reporting and Counting Geometric Intersections." In: *IEEE Trans. Comput.* 28.9 (Sept. 1979), pp. 643–647. ISSN: 0018-9340.
- [6] Herbert Edelsbrunner. "A new approach to rectangle intersections." In: *International Journal of Computer Mathematics* 13.3-4 (1983), pp. 221–229.
- [7] Herbert Edelsbrunner. "A new approach to rectangle intersections part I." In: *International Journal of Computer Mathematics* 13.3-4 (1983), pp. 209–219.
- [8] T. A. Nguyen, W. A. Perkins, T. J. Laffey, and D. Pecora. "Checking an Expert Systems Knowledge Base for Consistency and Completeness." In: *Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 1. IJCAI'85*. Los Angeles, California: Morgan Kaufmann Publishers Inc., 1985, pp. 375–378.
- [9] Tin A Nguyen, Walton Perkin, Thomas J Laffey, and Deanne Pecora. "Knowledge Base Verification." In: *AI Mag.* 8.2 (June 1987), pp. 69–75. ISSN: 0738-4602.
- [10] Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML*. Cambridge, MA, USA: MIT Press, 1990. ISBN: 0262132559.
- [11] Kurt Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Vol. 1, Basic Concepts*. English. 2. printing with few corrections 1997. ISBN: 3-540-60943-1. Springer, 1992. ISBN: 3-540-55597-8.
- [12] J. Vanthienen and E. Dries. *Developments in decision tables: evolution, applications and a proposed standard*. Research Report 9227. Leuven, Belgium: KU Leuven, 1992.

- [13] Thomas H. Davenport. *Process Innovation: Reengineering Work Through Information Technology*. Boston, MA, USA: Harvard Business School Press, 1993. ISBN: 0-87584-366-2.
- [14] J. Vanthienen and G. Wets. "Integration of the Decision Table Formalism with a Relational Database Environment." In: *Inf. Syst.* 20.7 (Nov. 1995), pp. 595–616.
- [15] Wil M. P. van der Aalst. "Verification of Workflow Nets." In: *Proceedings of the 18th International Conference on Application and Theory of Petri Nets*. ICATPN '97. Berlin, Heidelberg: Springer-Verlag, 1997, pp. 407–426. ISBN: 3-540-63139-9.
- [16] Wil M. P. van der Aalst. "The Application of Petri Nets to Workflow Management." In: *Journal of Circuits, Systems, and Computers* 8.1 (1998), pp. 21–66.
- [17] Edmund M. Clarke Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. Cambridge, MA, USA: MIT Press, 1999. ISBN: 0-262-03270-8.
- [18] Coverpages.com. *Simple Rule Markup Language (SRML)*. 2001. URL: <http://xml.coverpages.org/srml.html> (visited on 03/27/2019).
- [19] Juliane Dehnert and Peter Rittgen. "Relaxed Soundness of Business Processes." In: *Advanced Information Systems Engineering*. Ed. by Klaus R. Dittrich, Andreas Geppert, and Moira C. Norrie. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 157–170. ISBN: 978-3-540-45341-3.
- [20] *Workflow Management: Models, Methods, and Systems*. Cambridge, MA, USA: MIT Press, 2002. ISBN: 0262011891.
- [21] Anne Vinter Ratzner, Lisa Wells, Henry Michael Lassen, Mads Laursen, Jacob Frank Qvortrup, Martin Stig Stissing, Michael Westergaard, Søren Christensen, and Kurt Jensen. "CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets." In: *Applications and Theory of Petri Nets 2003*. Ed. by Wil M. P. van der Aalst and Eike Best. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 450–462. ISBN: 978-3-540-44919-5.
- [22] Shazia Sadiq, Maria Orłowska, Wasim Sadiq, and Cameron Foulger. "Data Flow and Validation in Workflow Modelling." In: *Proceedings of the 15th Australasian Database Conference - Volume 27*. ADC '04. Dunedin, New Zealand: Australian Computer Society, Inc., 2004, pp. 207–214.
- [23] R.A. Toorn, van der. "Component-based software design with Petri nets : an approach based on inheritance of behavior." PhD thesis. Department of Mathematics and Computer Science, 2004. ISBN: 90-386-0802-0.

- [24] Axel Martens. "Analyzing Web Service Based Business Processes." In: *Fundamental Approaches to Software Engineering*. Ed. by Maura Cerioli. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 19–33. ISBN: 978-3-540-31984-9.
- [25] Frank Puhmann and Mathias Weske. "Interaction Soundness for Service Orchestrations." In: *Service-Oriented Computing – ICSOC 2006*. Ed. by Asit Dan and Winfried Lamersdorf. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 302–313. ISBN: 978-3-540-68148-9.
- [26] Jan Recker, Marta Indulska, Michael Rosemann, and Peter F. Green. "How good is BPMN really? Insights from theory and practice." In: *ECIS*. 2006, pp. 1582–1593.
- [27] Ahmed Awad. "BPMN-Q: A Language to Query Business Processes." In: *Enterprise Modelling and Information Systems Architectures - Concepts and Applications , Proceedings of the 2nd International Workshop on Enterprise Modelling and Information Systems Architectures (EMISA'07), St. Goar, Germany, October 8-9, 2007*. Ed. by Manfred Reichert, Stefan Strecker, and Klaus Turowski. Vol. P-119. LNI. GI, 2007, pp. 115–128.
- [28] James Taylor and Neil Raden. *Smart (Enough) Systems: How to Deliver Competitive Advantage by Automating the Decisions Hidden in Your Business*. First. Upper Saddle River, NJ, USA: Prentice Hall Press, 2007. ISBN: 9780132347969.
- [29] Ahmed Awad, Gero Decker, and Mathias Weske. "Efficient Compliance Checking Using BPMN-Q and Temporal Logic." In: *Business Process Management*. Ed. by Marlon Dumas, Manfred Reichert, and Ming-Chien Shan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 326–341.
- [30] Remco M. Dijkman, Marlon Dumas, and Chun Ouyang. "Semantics and Analysis of Business Process Models in BPMN." In: *Inf. Softw. Technol.* 50.12 (Nov. 2008), pp. 1281–1294. ISSN: 0950-5849.
- [31] Marwane El Kharbili, Ana Karla A. de Medeiros, Sebastian Stein, and Wil M. P. van der Aalst. "Business Process Compliance Checking: Current State and Future Challenges." In: *MobIS*. Vol. 141. LNI. GI, 2008, pp. 107–113.
- [32] Michael zur Muehlen and Jan Recker. "How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation." In: *Advanced Information Systems Engineering*. Ed. by Zohra Bellahsene and Michel Léonard. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 465–479. ISBN: 978-3-540-69534-9.

- [33] Ahmed Awad, Matthias Weidlich, and Mathias Weske. "Specification, Verification and Explanation of Violation for Data Aware Compliance Rules." In: *Service-Oriented Computing*. Ed. by Luciano Baresi, Chi-Hung Chi, and Jun Suzuki. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 500–515. ISBN: 978-3-642-10383-4.
- [34] Volker Gruhn and Ralf Laue. "Reducing the cognitive complexity of business process models." In: *IEEE ICCI*. IEEE Computer Society, 2009, pp. 339–345.
- [35] Barbara von Halle and Larry Goldberg. *The Decision Model: A Business Logic Framework Linking Business and Technology*. 1st. Boston, MA, USA: Auerbach Publications, 2009. ISBN: 97814200-82814.
- [36] Kurt Jensen and Lars M. Kristensen. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. 1st. Springer Publishing Company, Incorporated, 2009. ISBN: 9783642002830.
- [37] David Knuplesch, Linh Thao Ly, Stefanie Rinderle-Ma, Holger Pfeifer, and Peter Dadam. "On Enabling Data-Aware Compliance Checking of Business Process Models." In: *Conceptual Modeling – ER 2010*. Ed. by Jeffrey Parsons, Motoshi Saeki, Peretz Shoval, Carson Woo, and Yair Wand. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 332–346. ISBN: 978-3-642-16373-9.
- [38] David Knuplesch, Linh Thao Ly, Stefanie Rinderle-Ma, Holger Pfeifer, and Peter Dadam. "On Enabling Data-Aware Compliance Checking of Business Process Models." In: *Conceptual Modeling – ER 2010*. Ed. by Jeffrey Parsons, Motoshi Saeki, Peretz Shoval, Carson Woo, and Yair Wand. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 332–346. ISBN: 978-3-642-16373-9.
- [39] Elena Kornysheva and Rébecca Deneckère. "Decision-Making Ontology for Information System Engineering." In: *Conceptual Modeling – ER 2010*. Ed. by Jeffrey Parsons, Motoshi Saeki, Peretz Shoval, Carson Woo, and Yair Wand. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 104–117. ISBN: 978-3-642-16373-9.
- [40] Hamed Mili, Guy Tremblay, Guitta Bou Jaoude, Éric Lefebvre, Lamia Elabed, and Ghizlane El Boussaidi. "Business Process Modeling Languages: Sorting Through the Alphabet Soup." In: *ACM Comput. Surv.* 43.1 (Dec. 2010), 4:1–4:56. ISSN: 0360-0300.
- [41] Michael zur Muehlen and Marta Indulska. "Modeling languages for business processes and business rules: A representational analysis." In: *Information Systems* 35.4 (2010). Vocabular-

- ies, Ontologies and Rules for Enterprise and Business Process Modeling and Management, pp. 379–390. ISSN: 0306-4379.
- [42] Wolfgang Reisig. *Petrinetze: Modellierungstechnik, Analysemethoden, Fallstudien*. Studium. Wiesbaden: Vieweg + Teubner, 2010. ISBN: 978-3-8348-1290-2.
- [43] Natalia Sidorova, Christian Stahl, and Nikola Trčka. “Workflow Soundness Revisited: Checking Correctness in the Presence of Data While Staying Conceptual.” In: *Advanced Information Systems Engineering*. Ed. by Barbara Pernici. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 530–544. ISBN: 978-3-642-13094-6.
- [44] W. M. P. van der Aalst, K. M. van Hee, A. H. M. ter Hofstede, N. Sidorova, H. M. W. Verbeek, M. Voorhoeve, and M. T. Wynn. “Soundness of workflow nets: classification, decidability, and analysis.” In: *Formal Aspects of Computing* 23.3 (May 2011), pp. 333–363.
- [45] Wil van der Aalst and Christian Stahl. *Modeling Business Processes: A Petri Net-Oriented Approach*. The MIT Press, 2011. ISBN: 9780262015387.
- [46] Jörg Becker, Martin Kugeler, and Michael Rosemann. *Process Management: A Guide for the Design of Business Processes*. 2nd. Springer Publishing Company, Incorporated, 2011. ISBN: 9783642151897.
- [47] Ran Cheng, Shazia Sadiq, and Marta Indulska. “Framework for Business Process and Rule Integration: A Case of BPMN and SBVR.” In: *Business Information Systems*. Ed. by Witold Abramowicz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 13–24. ISBN: 978-3-642-21863-7.
- [48] Jörg Hohwiller, Diethelm Schlegel, Gunter Grieser, and Yvette Hoekstra. “Integration of BPM and BRM.” In: *Business Process Model and Notation*. Ed. by Remco Dijkman, Jörg Hofstetter, and Jana Koehler. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 136–141. ISBN: 978-3-642-25160-3.
- [49] Natalia Sidorova, Christian Stahl, and Nikola Trčka. “Soundness verification for conceptual workflow nets with data: Early detection of errors with the most precision possible.” In: *Information Systems* 36.7 (2011). Special Issue: Advanced Information Systems Engineering (CAiSE’10), pp. 1026–1043. ISSN: 0306-4379.
- [50] Martijn Zoet, Johan Versendaal, Pascal Ravesteyn, and Richard J. Welke. “Alignment of business process management and business rules.” In: *ECIS*. 2011.

- [51] Cédric Favre and Hagen Völzer. "The Difficulty of Replacing an Inclusive OR-Join." In: *Business Process Management*. Ed. by Alistair Barros, Avigdor Gal, and Ekkart Kindler. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 156–171. ISBN: 978-3-642-32885-5.
- [52] Alan N. Fish. *Knowledge automation: how to implement decision management in business processes*. USA: John Wiley & Sons, 2012. ISBN: 9781118094761.
- [53] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. 2nd. Springer Publishing Company, Incorporated, 2012. ISBN: 9783642286155.
- [54] A. Zarghami, B. Sapkota, M. Z. Eslami, and M. van Sinderen. "Decision as a Service: Separating Decision-making from Application Process Logic." In: *2012 IEEE 16th International Enterprise Distributed Object Computing Conference*. Sept. 2012, pp. 103–112.
- [55] Diego Calvanese, Giuseppe De Giacomo, and Marco Montali. "Foundations of Data-aware Process Analysis: A Database Theory Perspective." In: *Proceedings of the 32Nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. PODS '13. New York, New York, USA: ACM, 2013, pp. 1–12. ISBN: 978-1-4503-2066-5.
- [56] Michael Totschnig, Christian Willems, and Christoph Meinel. "openHPI: Evolution of a MOOC Platform from LMS to SOA." In: ed. by Owen Foley, Maria Teresa Restivo, James Onohuome Uhomoibhi, and Markus Helfert. SciTePress, 2013, pp. 593–598.
- [57] Semra Catalkaya, David Knuplesch, Carolina Chiao, and Manfred Reichert. "Enriching Business Process Models with Decision Rules." In: *Business Process Management Workshops*. Ed. by Niels Lohmann, Minseok Song, and Petia Wohed. Cham: Springer International Publishing, 2014, pp. 198–211. ISBN: 978-3-319-06257-0.
- [58] Tom Debevoise and James Taylor. *The MicroGuide to Process and Decision Modeling in BPMN/DMN: Building More Effective Processes by Integrating Process Modeling with Decision Modeling*. USA: CreateSpace Independent Publishing Platform, 2014. ISBN: 9781502789648.
- [59] OMG. *Business Process Model and Notation, Version 2.0.2*. OMG Standard. Jan. 2014.
- [60] Kimon Batoulis, Andreas Meyer, Ekaterina Bazhenova, Gero Decker, and Mathias Weske. "Extracting Decision Logic from Process Models." In: *Advanced Information Systems Engineering*.

- Ed. by Jelena Zdravkovic, Marite Kirikova, and Paul Johanneson. Cham: Springer International Publishing, 2015, pp. 349–366. ISBN: 978-3-319-19069-3.
- [61] OMG. *Decision Model and Notation, Version 1.0*. OMG Standard. Sept. 2015.
- [62] Han van der Aa, Henrik Leopold, Kimon Batoulis, Mathias Weske, and Hajo A. Reijers. “Integrated Process and Decision Modeling for Data-Driven Processes.” In: *Business Process Management Workshops*. Ed. by Manfred Reichert and Hajo A. Reijers. Cham: Springer International Publishing, 2016, pp. 405–417. ISBN: 978-3-319-42887-1.
- [63] Kimon Batoulis, Anne Baumgraß, Nico Herzberg, and Mathias Weske. “Enabling Dynamic Decision Making in Business Processes with DMN.” In: *Business Process Management Workshops*. Ed. by Manfred Reichert and Hajo A. Reijers. Cham: Springer International Publishing, 2016, pp. 418–431. ISBN: 978-3-319-42887-1.
- [64] Diego Calvanese, Marlon Dumas, Üleri Laurson, Fabrizio M. Maggi, Marco Montali, and Irene Teinemaa. “Semantics and Analysis of DMN Decision Tables.” In: *Business Process Management*. Ed. by Marcello La Rosa, Peter Loos, and Oscar Pastor. Cham: Springer International Publishing, 2016, pp. 217–233. ISBN: 978-3-319-45348-4.
- [65] Carlo Combi, Barbara Oliboni, Alessandro Zardiniy, and Francesca Zerbato. “Seamless Design of Decision-Intensive Care Pathways.” In: *ICHI*. IEEE Computer Society, 2016, pp. 35–45.
- [66] Riadh Ghlala, Zahra Kodja Aouina, and Lamjed Ben Said. “BPMN Decision Footprint: Towards Decision Harmony Along BI Process.” In: *Information and Software Technologies*. Ed. by Giedre Dregvaite and Robertas Damasevicius. Cham: Springer International Publishing, 2016, pp. 269–284. ISBN: 978-3-319-46254-7.
- [67] Paul Harmon. *The State of Business Process Management 2016*. Tech. rep. BPTrends, 2016.
- [68] Knut Hinkelmann. “Business Process Flexibility and Decision-Aware Modeling—The Knowledge Work Designer.” In: *Domain-Specific Conceptual Modeling: Concepts, Methods and Tools*. Ed. by Dimitris Karagiannis, Heinrich C. Mayr, and John Mylopoulos. Cham: Springer International Publishing, 2016, pp. 397–414.
- [69] Laurent Janssens, Ekaterina Bazhenova, Johannes De Smedt, Jan Vanthienen, and Marc Denecker. “Consistent Integration of Decision (DMN) and Process (BPMN) Models.” In: *CAiSE Forum*. Vol. 1612. CEUR Workshop Proceedings. CEUR-WS.org, 2016, pp. 121–128.

- [70] Laurent Janssens, Johannes De Smedt, and Jan Vanthienen. "Modeling and Enacting Enterprise Decisions." In: *Advanced Information Systems Engineering Workshops*. Ed. by John Krogstie, Haralambos Mouratidis, and Jianwen Su. Cham: Springer International Publishing, 2016, pp. 169–180. ISBN: 978-3-319-39564-7.
- [71] Krzysztof Kluza and Krzysztof Honkisz. "From SBVR to BPMN and DMN Models. Proposal of Translation from Rules to Process and Decision Models." In: *Artificial Intelligence and Soft Computing*. Ed. by Leszek Rutkowski, Marcin Korytkowski, Rafał Scherer, Ryszard Tadeusiewicz, Lotfi A. Zadeh, and Jacek M. Zurada. Cham: Springer International Publishing, 2016, pp. 453–462. ISBN: 978-3-319-39384-1.
- [72] Matthias Kunze and Mathias Weske. *Behavioural Models - From Modelling Finite Automata to Analysing Business Processes*. Springer, 2016. ISBN: 978-3-319-44958-6.
- [73] Üleri Laurson and Fabrizio Maria Maggi. "A Tool for the Analysis of DMN Decision Tables." In: *Proceedings of the BPM Demo Track 2016 Co-located with the 14th International Conference on Business Process Management (BPM 2016), Rio de Janeiro, Brazil, September 21, 2016*. 2016, pp. 56–60.
- [74] Marco Montali and Diego Calvanese. "Soundness of data-aware, case-centric processes." In: *International Journal on Software Tools for Technology Transfer* 18.5 (Oct. 2016), pp. 535–558. ISSN: 1433-2787.
- [75] Adriatik Nikaj, Kimon Batoulis, and Mathias Weske. "REST-Enabled Decision Making in Business Process Choreographies." In: *Service-Oriented Computing*. Ed. by Quan Z. Sheng, Eleni Stroulia, Samir Tata, and Sami Bhiri. Cham: Springer International Publishing, 2016, pp. 547–554. ISBN: 978-3-319-46295-0.
- [76] OMG. *Decision Model and Notation, Version 1.1*. OMG Standard. June 2016.
- [77] Kimon Batoulis, Stephan Haarmann, and Mathias Weske. "Various Notions of Soundness for Decision-Aware Business Processes." In: *Conceptual Modeling*. Ed. by Heinrich C. Mayr, Giancarlo Guizzardi, Hui Ma, and Oscar Pastor. Cham: Springer International Publishing, 2017, pp. 403–418. ISBN: 978-3-319-69904-2.
- [78] Kimon Batoulis, Alexey Nesterenko, Guenther Repitsch, and Mathias Weske. "Decision Management in the Insurance Industry: Standards and Tools." In: *Proceedings of the BPM 2017 Industry Track co-located with the 15th International Conference on Business Process Management (BPM 2017), Barcelona, Spain, September 10-15, 2017*. 2017, pp. 52–63.

- [79] Kimon Batoulis and Mathias Weske. "A Tool for Checking Soundness of Decision-Aware Business Processes." In: *Proceedings of the BPM Demo Track and BPM Dissertation Award co-located with 15th International Conference on Business Process Modeling (BPM 2017), Barcelona, Spain, September 13, 2017*. 2017.
- [80] Kimon Batoulis and Mathias Weske. "Soundness of Decision-Aware Business Processes." In: *Business Process Management Forum*. Ed. by Josep Carmona, Gregor Engels, and Akhil Kumar. Cham: Springer International Publishing, 2017, pp. 106–124. ISBN: 978-3-319-65015-9.
- [81] Joachim Bossuyt and Frederik Gailly. *Investigating The Benefits Of Modeling Business Processes With Bpmn + Dmn*. Gent, Belgium, 2017.
- [82] Diego Calvanese, Marlon Dumas, Fabrizio M. Maggi, and Marco Montali. "Semantic DMN: Formalizing Decision Models with Domain Knowledge." In: *Rules and Reasoning*. Ed. by Stefania Costantini, Enrico Franconi, William Van Woensel, Roman Kontchakov, Fariba Sadri, and Dumitru Roman. Cham: Springer International Publishing, 2017, pp. 70–86. ISBN: 978-3-319-61252-2.
- [83] Carlo Combi, Barbara Oliboni, Alessandro Zardini, and Francesca Zerbato. "A Methodological Framework for the Integrated Design of Decision-Intensive Care Pathways—an Application to the Management of COPD Patients." In: *Journal of Healthcare Informatics Research* 1.2 (Dec. 2017), pp. 157–217. ISSN: 2509-498X.
- [84] Johannes De Smedt, Seppe K. L. M. vanden Broucke, Josue Obregon, Aekyung Kim, Jae-Yoon Jung, and Jan Vanthienen. "Decision Mining in a Broader Context: An Overview of the Current Landscape and Future Directions." In: *Business Process Management Workshops*. Ed. by Marlon Dumas and Marcelo Fantinato. Cham: Springer International Publishing, 2017, pp. 197–207. ISBN: 978-3-319-58457-7.
- [85] Johannes De Smedt, Faruk Hasić, Seppe K. L. M. vanden Broucke, and Jan Vanthienen. "Towards a Holistic Discovery of Decisions in Process-Aware Information Systems." In: *Business Process Management*. Ed. by Josep Carmona, Gregor Engels, and Akhil Kumar. Cham: Springer International Publishing, 2017, pp. 183–199. ISBN: 978-3-319-65000-5.
- [86] Faruk Hasić, Johannes De Smedt, and Jan Vanthienen. "A Service-Oriented Architecture Design of Decision-Aware Information Systems: Decision as a Service." In: *On the Move to Meaningful Internet Systems. OTM 2017 Conferences*. Ed. by Hervé Panetto, Christophe Debruyne, Walid Gaaloul, Mike Papazoglou, Adrian Paschke, Claudio Agostino Ardagna, and Robert

- Meersman. Cham: Springer International Publishing, 2017, pp. 353–361. ISBN: 978-3-319-69462-7.
- [87] Faruk Hasić, Johannes De Smedt, and Jan Vanthienen. “Towards Assessing the Theoretical Complexity of the Decision Model and Notation (DMN).” In: *RADAR+EMISA@CAiSE*. Vol. 1859. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 64–71.
- [88] Faruk Hasić, Linus Vanwijck, and Jan Vanthienen. “Integrating Processes, Cases, and Decisions for Knowledge-Intensive Process Modelling.” In: *Proceedings of the 1st International Workshop on Practicing Open Enterprise Modeling within OMiLAB (PrOse 2017) co-located with 10th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modelling (PoEM 2017), Leuven, Belgium, November 22, 2017*. 2017.
- [89] Florian Imgrund, Mathäus Malorny, and Christian Janiesch. “Eine Literaturanalyse zur Integration von Business Rules und Business Process Management.” In: *Wirtschaftsinformatik*. Ed. by Jan Marco Leimeister and Walter Brenner. 2017.
- [90] Mathias Kirchmer. *High Performance through Business Process Management. Strategy Execution in a Digital World*. 3rd. Springer, 2017. ISBN: 9783319512587.
- [91] Krzysztof Kluza and Grzegorz J. Nalepa. “A method for generation and design of business processes with business rules.” In: *Information and Software Technology* 91 (2017), pp. 123–141. ISSN: 0950-5849.
- [92] David Knuplesch and Manfred Reichert. “A Visual Language for Modeling Multiple Perspectives of Business Process Compliance Rules.” In: *Softw. Syst. Model.* 16.3 (July 2017), pp. 715–736. ISSN: 1619-1366.
- [93] OMG. *Semantics of Business Vocabulary and Rules, Version 1.4*. OMG Standard. May 2017.
- [94] Luise Pufahl, Sankalita Mandal, Kimon Batoulis, and Mathias Weske. “Re-evaluation of Decisions Based on Events.” In: *Enterprise, Business-Process and Information Systems Modeling*. Ed. by Iris Reinhartz-Berger, Jens Gulden, Selmin Nurcan, Wided Guédria, and Palash Bera. Cham: Springer International Publishing, 2017, pp. 68–84. ISBN: 978-3-319-59466-8.
- [95] Kimon Batoulis and Mathias Weske. “A Tool for the Uniqueification of DMN Decision Tables.” In: *Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2018 co-located with 16th International Conference on Business Process Management (BPM 2018), Sydney, Australia, September 9-14, 2018*. 2018, pp. 116–119.

- [96] Kimon Batoulis and Mathias Weske. "Disambiguation of DMN Decision Tables." In: *Business Information Systems*. Ed. by Witold Abramowicz and Adrian Paschke. Cham: Springer International Publishing, 2018, pp. 236–249. ISBN: 978-3-319-93931-5.
- [97] Diego Calvanese, Marlon Dumas, Üleri Laurson, Fabrizio M. Maggi, Marco Montali, and Irene Teinemaa. "Semantics, Analysis and Simplification of DMN Decision Tables." In: *Information Systems* (2018). ISSN: 0306-4379.
- [98] Carl Corea and Patrick Delfmann. "A Tool to Monitor Consistent Decision-Making in Business Process Execution." In: *BPM (Dissertation/Demos/Industry)*. Vol. 2196. CEUR Workshop Proceedings. CEUR-WS.org, 2018, pp. 76–80.
- [99] Carl Corea and Patrick Delfmann. "Supporting Business Rule Management with Inconsistency Analysis." In: *BPM (Dissertation/Demos/Industry)*. Vol. 2196. CEUR Workshop Proceedings. CEUR-WS.org, 2018, pp. 141–147.
- [100] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management, Second Edition*. Springer, 2018.
- [101] Stephan Haarmann, Kimon Batoulis, and Mathias Weske. "Compliance Checking for Decision-Aware Process Models." In: *Business Process Management Workshops - BPM 2018 International Workshops, Sydney, NSW, Australia, September 9-14, 2018, Revised Papers*. 2018, pp. 494–506.
- [102] Faruk Hasić, Lesly Devadder, Maxim Dochez, Jonas Hanot, Johannes De Smedt, and Jan Vanthienen. "Challenges in Refactoring Processes to Include Decision Modelling." In: *Business Process Management Workshops*. Ed. by Ernest Teniente and Matthias Weidlich. Cham: Springer International Publishing, 2018, pp. 529–541. ISBN: 978-3-319-74030-0.
- [103] Faruk Hasić, Johannes De Smedt, and Jan Vanthienen. "Augmenting processes with decision intelligence: Principles for integrated modelling." In: *Decision Support Systems* 107 (2018), pp. 1–12. ISSN: 0167-9236.
- [104] Faruk Hasić, Johannes De Smedt, and Jan Vanthienen. "Re-designing Processes for Decision-Awareness: Strategies for Integrated Modelling." In: *QUATIC*. IEEE Computer Society, 2018, pp. 247–250.
- [105] Krzysztof Kluza and Grzegorz J. Nalepa. "Formal Model of Business Processes Integrated with Business Rules." In: *Information Systems Frontiers* (Feb. 2018). ISSN: 1572-9419.

- [106] Massimiliano de Leoni, Paolo Felli, and Marco Montali. "A Holistic Approach for Soundness Verification of Decision-Aware Process Models." In: *Conceptual Modeling*. Ed. by Juan C. Trujillo, Karen C. Davis, Xiaoyong Du, Zhanhuai Li, Tok Wang Ling, Guoliang Li, and Mong Li Lee. Cham: Springer International Publishing, 2018, pp. 219–235. ISBN: 978-3-030-00847-5.
- [107] Massimiliano de Leoni, Paolo Felli, and Marco Montali. "A Holistic Approach for Soundness Verification of Decision-Aware Process Models (extended version)." In: *CoRR abs/1804.02316* (2018).
- [108] F. Mannhardt. "Multi-perspective process mining." English. Proefschrift. PhD thesis. Department of Mathematics and Computer Science, Feb. 2018. ISBN: 978-90-386-4438-7.
- [109] Roberto Posenato, Francesca Zerbato, and Carlo Combi. "Managing Decision Tasks and Events in Time-Aware Business Process Models." In: *Business Process Management*. Ed. by Mathias Weske, Marco Montali, Ingo Weber, and Jan vom Brocke. Cham: Springer International Publishing, 2018, pp. 102–118. ISBN: 978-3-319-98648-7.
- [110] Ekaterina Bazhenova, Francesca Zerbato, Barbara Oliboni, and Mathias Weske. "From BPMN process models to DMN decision models." In: *Information Systems* 83 (2019), pp. 69–88. ISSN: 0306-4379.
- [111] Faruk Hasić, Alexander De Craemer, Thijs Hegge, Gideon Magala, and Jan Vanthienen. "Measuring the Complexity of DMN Decision Models." In: *Business Process Management Workshops*. Ed. by Florian Daniel, Quan Z. Sheng, and Hamid Motahari. Cham: Springer International Publishing, 2019, pp. 514–526. ISBN: 978-3-030-11641-5.
- [112] Faruk Hasić and Jan Vanthienen. "Complexity metrics for DMN decision models." In: *Computer Standards & Interfaces* (2019). ISSN: 0920-5489.
- [113] Sabine Nagel, Carl Corea, and Patrick Delfmann. "Effects of Quantitative Measures on Understanding Inconsistencies in Business Rules." In: *Proceedings of the 52nd Hawaii International Conference on System Sciences*. IEE, 2019.
- [114] OMG. *Decision Model and Notation, Version 1.2*. OMG Standard. Jan. 2019.
- [115] Suchenia, Anna, Kluza, Krzysztof, Wiśniewski, Piotr, Jobczyk, Krystian, and Ligeza, Antoni. "Towards knowledge interoperability between the UML, DMN, BPMN and CMMN models." In: *MATEC Web Conf.* 252 (2019), p. 02011.

- [116] Michael zur Muehlen, Marta Indulska, and Kai Kittel. "Towards Integrated Modeling of Business Processes and Business Rules." In: *ACIS 2008 Proceedings*. 108. 208.

