AUTOMATENBASIERTE VERGLEICHSMETHODEN UND VERHALTENSANALYSEVERFAHREN FÜR IMPERATIVE UND DEKLARATIVE PROZESSMODELLE

NICOLAI SCHÜTZENMEIER

DISSERTATION

UNIVERSITÄT BAYREUTH

INSTITUT FÜR INFORMATIK
FAKULTÄT FÜR MATHEMATIK, PHYSIK, INFORMATIK

Nicolai Schützenmeier: *Automatenbasierte Vergleichsmethoden und Verhaltensanalyseverfahren für imperative und deklarative Prozessmodelle*, Dissertation, © 2024

Universität Bayreuth
Institut für Informatik

Automatenbasierte Vergleichsmethoden und Verhaltensanalyseverfahren für imperative und deklarative Prozessmodelle

Von der Universität Bayreuth zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Abhandlung

von
Nicolai Tobias Schützenmeier
geboren in Marktredwitz

1. Gutachter: Prof. Dr.-Ing. Stefan Jablonski
2. Gutachter: Prof. Dr. Maximilian Röglinger

Tag der Einreichung: 10.01.2024
Tag des Kolloquiums: 25.03.2024

Zusammenfassung

Geschäftsprozessmanagement gilt als systematischer Ansatz, um sowohl automatisierte als auch nichtautomatisierte Prozesse unter anderem zu erfassen, zu gestalten, auszuführen, zu dokumentieren und zu steuern und damit nachhaltig die mit der Unternehmensstrategie abgestimmten Ziele zu erreichen. Dabei wird grundsätzlich zwischen zwei Arten von Prozessen unterschieden. Zum einen gibt es die sogenannten Routineprozesse. Dabei handelt es sich um Prozesse, die immer nach einem gewissen Schema oder Muster ablaufen und die folglich sehr wenig Flexibilität in der Ausführung aufweisen. Diese Art von Prozessen wird mit Hilfe von sogenannten imperativen Prozessmodellierungssprachen dargestellt. Auf der anderen Seite kennt man die sogenannten flexiblen Prozesse, die sehr viele Freiheiten und demzufolge sehr viele Ausführungsmöglichkeiten besitzen. Für diese Art von Prozessen existiert ebenfalls ein Modellierungsparadigma, die sogenannte deklarative Prozessmodellierung. Im Laufe der Zeit haben sich auf Grund verschiedenster Anforderungen für beide Paradigmen zahlreiche Modellierungssprachen entwickelt.

Durch die Verwendung verschiedener Modellierungssprachen treten bei der Fusionierung unterschiedlicher Unternehmen, beim Überarbeiten oder Anpassen von Prozessmodellen immer wieder Probleme und Unklarheiten auf. So ist es in der Praxis oftmals eine ziemliche Herausforderung, die jeweiligen unternehmensspezifischen Prozesse bzw. Prozessmodelle miteinander zu kombinieren. Gerade die Verwendung deklarativer Prozessmodellierungssprachen hat aufgezeigt, dass auf Grund der hohen Komplexität deklarativer Prozessmodelle derartige Aufgaben ohne Softwareunterstützung kaum zu handhaben sind.

Die Forschung dieser Dissertation widmet sich obig beschriebenen Problemstellungen, indem automatenbasierte Lösungen zum Vergleich von sowohl imperativen als auch deklarativen Prozessmodellen entwickelt werden. Des Weiteren werden Möglichkeiten vorgestellt, die es erlauben, aus verschiedenen Prozessmodellen Gemeinsamkeiten, Unterschiede und Ähnlichkeiten herauszuarbeiten. Diese können in der Praxis genutzt werden, um eine erfolgreiche Fusionierung verschiedener Prozesse zu erleichtern. Darüber hinaus wird die Verständlichkeit deklarativer Prozessmodelle durch das Entwickeln und Implementieren verschiedener Verhaltensanalysen immens verbessert, was eine durchgeführte Benutzerstudie zeigt.

Abstract

Business Process Management is regarded as a systematic approach to capture, design, execute, document, and control both automated and non-automated processes, with the ultimate goal of achieving objectives aligned with the company's strategic vision. Basically two fundamental types of processes are distinguished. First of all, there are those known as routine processes. These are processes that consistently follow a specific pattern or template, resulting in very limited flexibility in execution. These processes are represented with the help of imperative process modeling languages. On the other hand, there are those called flexible processes. These processes enjoy a high degree of flexibility, offering numerous execution possibilities. For these processes, a modeling paradigm known as declarative process modeling is used. Over time, due to various requirements, numerous modeling languages have evolved for both paradigms.

Sometimes the use of various modeling languages gives rise to problems and ambiguities, especially when different companies merge or process models are revised or adapted. Thus, in practice, it is often a challenging task to integrate the respective company-specific processes or process models. The use of declarative process modeling languages has revealed that, due to the complexity of declarative process models, such tasks are hardly manageable without software support.

The research conducted in this dissertation is dedicated to the aforementioned issues by developing automated solutions for comparing both imperative and declarative process models. Furthermore, it presents methods for identifying commonalities, differences, and similarities among different process models. These findings can be practically applied to facilitate the successful merger of various processes. Last but not least, the comprehensibility of declarative process models is significantly enhanced through the elaboration and implementation of various behavioral analyses, as indicated by a conducted user study.

Danksagung

Ein Projekt wie diese Dissertation kann nur mit tatkräftiger Unterstützung von vielen Seiten verwirklicht werden. An dieser Stelle möchte ich mich daher bei all jenen Personen bedanken, die mir oft auf ihre ganz eigene Art und Weise bei der Anfertigung dieser Arbeit geholfen haben.

Zunächst möchte ich meinem Doktorvater, Prof. Dr.-Ing. Stefan Jablonski, einen besonderen Dank aussprechen. Er hat mir mit der Anstellung als wissenschaftlicher Mitarbeiter an seinem Lehrstuhl für Angewandte Informatik IV an der Universität Bayreuth sein Vertrauen geschenkt und mir damit die Gelegenheit zur weiteren Forschung und letztendlich zur Promotion eröffnet. Während meiner gesamten Tätigkeit stand er mir jederzeit mit Rat und Tat, großer Geduld und konstruktiver Kritik hilfreich zur Seite. Ebenso möchte ich Herrn Prof. Dr. Maximilian Röglinger für die Übernahme des Zweitgutachtens danken.

Ein aufrichtiger Dank gilt auch all meinen Kolleginnen und Kollegen am Lehrstuhl für die jahrelange harmonische Teamarbeit, die mich persönlich positiv geprägt hat. Namentlich hervorheben möchte ich Dr. Lars Ackermann, Christian Sturm, Myriel Fichtner, Philipp Eisenhuth, Julian Neuberger, Sebastian Petter, Martin Käppel und Prof. Dr. Stefan Schönig, zu denen während meiner Zeit am Lehrstuhl ein freundschaftliches Verhältnis entstanden ist. Zusammen haben wir zahlreiche Fachbeiträge erarbeitet, über deren Veröffentlichung wir uns gemeinsam freuen konnten. Zudem hatten meine Kolleginnen und Kollegen stets ein offenes Ohr für meine Fragen und Anliegen. In den vielen produktiven Fachdiskussionen in freundschaftlicher Atmosphäre, die oft auch außerhalb des Büros stattfanden, haben sich viele interessante Gedanken, Anregungen und Literaturtipps ergeben, die sich später als wichtig und richtungsweisend für diese Dissertation erwiesen haben.

Außerdem gebührt mein aufrichtiger Dank Prof. Dr. Patrick Delfmann und Dr. Carl Corea von der Universität Koblenz für den wertvollen Austausch und die zahlreichen Diskussionen, die meine Forschung maßgeblich bereichert haben. Ihre konstruktiven Inputs haben mein Verständnis vertieft und neue Perspektiven eröffnet, die sogar zu einer gemeinsamen Publikation geführt haben.

Des Weiteren möchte ich lobende Worte an alle weiteren Mitarbeitenden an der Universität

Bayreuth richten. Ein herzliches Dankeschön gilt insbesondere unserer Sekretärin Kerstin Haseloff, die stets für einen reibungslosen Ablauf im bisweilen hektischen Lehrstuhlbetrieb gesorgt hat und uns bei behördlichen und verwaltungstechnischen Angelegenheiten unterstützt hat. Auch unserem Techniker Bernd Schlesier möchte ich danken, der immer zur Stelle war, wenn es um Hardwareanschaffungen oder technische Probleme ging. Weiterhin möchte ich allen Teilnehmerinnen und Teilnehmern meiner Benutzerstudie danken. Ihr Verständnis und die Zeit, die sie für die Durchführung, Rückmeldung und Evaluation meiner Studie aufgebracht haben, sind für mich von unschätzbarem Wert.

Nicht zuletzt möchte ich meinen Eltern und meinem Bruder Christopher für ihre Geduld, ihr Verständnis, ihren emotionalen Zuspruch und ihre Ermutigung danken. Sie haben mich auf meinem gesamten Lebensweg, während des Studiums und der Arbeit an dieser Dissertation, begleitet und unterstützt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Seit den frühen 1990er Jahren hat sich das Geschäftsprozessmanagement[1] zu einem unverzichtbaren Instrument entwickelt, um Firmen und Organisationen dabei zu unterstützen, Abläufe zu optimieren, die Effizienz zu steigern und letztendlich ihre Unternehmensziele besser, schneller und flexibler zu erreichen [1]. In einer Welt, die von ständigem Wandel und wachsendem Wettbewerbsdruck geprägt ist, gewinnt die Fähigkeit, Prozesse zu analysieren, anzupassen und zu überwachen, zunehmend an Bedeutung. Im Kern geht es beim Prozessmanagement darum, Unternehmensprozesse systematisch zu erfassen, zu dokumentieren, zu analysieren und kontinuierlich zu verbessern [2]. Prozessmodelle spielen hierbei eine zentrale Rolle [2]. Diese Modelle visualisieren die einzelnen Schritte, Ressourcen und Interaktionen, die bei der Durchführung eines Geschäftsprozesses involviert sind. Sie ermöglichen nicht nur eine klare Darstellung der Prozessabläufe, sondern auch eine tiefere Analyse, um Potenziale zur Optimierung zu identifizieren. Diese Modelle dienen als Kommunikationsmittel, um ein gemeinsames Verständnis für Prozesse innerhalb des Unternehmens zu schaffen. Sie erleichtern die Identifizierung von Engpässen, Redundanzen und ineffizienten Schritten, was wiederum zur gezielten Prozessoptimierung führt. Im Laufe der Zeit wurden auf Grund verschiedener Anwendungsfälle sowohl in Industrie als auch Forschung zahlreiche verschiedene Prozessmodellierungssprachen entwickelt. Diese werden grundlegend in *imperative* und *deklarative* Prozessmodellierungssprachen unterteilt [1, 3, 4].

Imperative Prozessmodellierungssprachen zeichnen sich durch die explizite Festlegung von Anweisungen und Schritten aus, die in einer bestimmten Reihenfolge abgearbeitet werden müssen. Hierbei wird genau vorgegeben, wie ein Prozess durchgeführt werden soll, wodurch eine detaillierte Kontrolle über den Ablauf ermöglicht wird. Diese Sprachen eignen sich gut für die Modellierung von klaren, strukturierten Abläufen und sogenannten *Routineprozessen* [3]. Allerdings stoßen imperative Prozessmodellierungssprachen in komplexen und sich dynamisch

[1] In dieser Arbeit werden die beiden Begriffe *Geschäftsprozess* und *Geschäftsprozessmanagement* mit *Prozess* bzw. *Prozessmanagement* abgekürzt.

ändernden Umgebungen an ihre Grenzen. Hier kommt die deklarative Prozessmodellierung ins Spiel. Deklarative Prozessmodellierungssprachen ermöglichen eine abstraktere Spezifikation von Geschäftsprozessen, indem sie sich auf das „Was“ und nicht auf das „Wie“ konzentrieren. Statt genauer Anweisungen legen sie Regeln, Bedingungen und Beziehungen zwischen verschiedenen Aktivitäten fest. Dies bietet Flexibilität und Anpassungsfähigkeit, da die Ausführung dem System überlassen wird, das auf Basis der deklarativen Regeln den Prozess gestaltet. Der Übergang von imperativer zu deklarativer Modellierung wird besonders relevant, wenn es darum geht, mit sich ändernden Anforderungen und komplexen, sogenannten *flexiblen Prozessen* [3] umzugehen. Aufgrund vielfältiger Anforderungen in der Prozessmodellierung haben sich im Laufe der Zeit sowohl imperative als auch deklarative Sprachen entwickelt. Diese Vielfalt ermöglicht es, den unterschiedlichen Anforderungen und Charakteristiken von Geschäftsprozessen gerecht zu werden und bietet eine breite Palette an Modellierungswerkzeugen für verschiedene Szenarien.

Prozessmodelle unterscheiden sich unter anderem in ihrer Darstellungsform. Es gibt zum einen graphische Prozessmodellierungssprachen wie *Business Process Model and Notation (BPMN)*[2] oder *Case Management Model and Notation (CMMN)*[3], die einen Geschäftsprozess durch das Zusammenspiel von graphischen Elementen wie Pfeilen oder Rechtecken darstellen. Darüber hinaus existieren textbasierte Prozessmodellierungssprachen wie *Declarative Process Intermediate Language (DPIL)* [5] oder *WS-Business Process Execution Language (BPEL)*[4], in denen Zusammenhänge zwischen den Elementen eines Geschäftsprozesses durch Programmcode oder durch (oftmals auf Basis einer formalen Logik) vordefinierte Bedingungen formuliert werden. Des Weiteren entstanden Prozessmodellierungssprachen wie z.B. *Declare* [6], die sowohl eine graphische als auch eine textuelle Darstellungsform besitzen.

Ein weiterer wesentlicher Unterschied bei den Prozessmodellierungssprachen ist deren Ausdrucksmächtigkeit. Jede von ihnen besitzt ihre eigenen Vor- und Nachteile und stößt in gewissen Situationen an ihre Grenzen [7]. Außerdem können komplexere Zusammenhänge in bestimmten Prozessmodellierungssprachen viel leichter und übersichtlicher zu formulieren sein als in anderen [8]. Insbesondere der letzte Aspekt macht es häufig sehr schwer oder ohne die Unterstützung von Software sogar unmöglich, gemeinsame Elemente von Modellen[5], die in verschiedenen Modellierungssprachen formuliert sind, zu identifizieren. Die vorliegende Arbeit widmet sich dieser Herausforderung und liefert einen Ansatz, um Geschäftsprozessmodelle, die in unterschiedlichen Sprachen modelliert wurden, zu vergleichen und deren Gemeinsamkeiten sowie Unterschiede herauszuarbeiten.

[2] `http://www.omg.org/spec/BPMN/2.0`, abgerufen am: 27. Dezember 2023

[3] `https://www.omg.org/spec/CMMN/1.1/`, abgerufen am: 27. Dezember 2023

[4] `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`, abgerufen am: 27. Dezember 2023

[5] In dieser Arbeit wird der Begriff *Modell* synonym für *Prozessmodell* verwendet. Sofern *Modell* in einem anderen Zusammenhang auftaucht, erfolgt stets eine explizite Erwähnung.

Die Notwendigkeit, Prozessmodelle zu vergleichen, erstreckt sich über verschiedene Szenarien. Bei Fusionen von Unternehmen beispielsweise ist ein Vergleich von Modellen in verschiedenen Sprachen unerlässlich, um Geschäftsprozesse effektiv zu fusionieren und eine nahtlose Integration zu gewährleisten (Abbildung 1.1). Ebenso ist bei Änderungen von Prozessen ein sorgfältiger Vergleich von Modellen erforderlich, um sicherzustellen, dass die vorgenommenen Änderungen korrekt und ohne negative Auswirkungen auf andere Teile des Unternehmens erfolgen. Dieser Ansatz trägt dazu bei, die Integrität und Effizienz der Geschäftsprozesse sicherzustellen, indem potenzielle Fehler oder Unstimmigkeiten frühzeitig erkannt und behoben werden können.

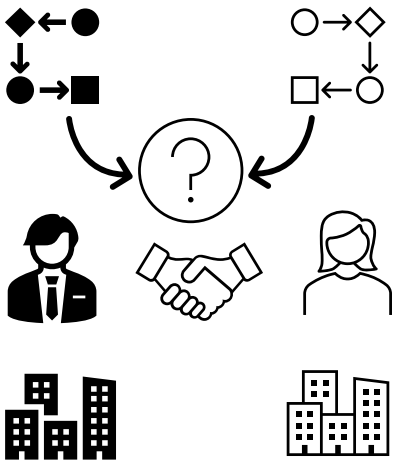


Abbildung 1.1: Notwendigkeit eines Prozessmodellvergleichs bei Fusion verschiedener Unternehmen

1.1 Motivation

Auf Grund des bereits erwähnten großen Angebots an verschiedenen Prozessmodellierungssprachen gibt es zu einem Prozess ebenso viele äquivalente Prozessmodelle. Da sich keine dieser Modellierungssprachen endgültig als Standard herauskristallisiert hat, werden teilweise sogar in verschiedenen Abteilungen eines Unternehmens unterschiedliche Modellierungssprachen verwendet. Damit ergeben sich natürlicherweise die folgenden Fragestellungen:

1. Sind zwei oder mehrere[6] (in verschiedenen Modellierungssprachen modellierte) Prozessmodelle gleich? Falls nein, ist eines der Modelle überspezifiziert? Worin bestehen die

[6]Im weiteren Verlauf dieser Arbeit wird stets von zwei zu vergleichenden Prozessmodellen gesprochen. Um den Vergleich mehrerer Prozessmodelle durchzuführen, können alle Methoden jeweils paarweise auf zwei Prozessmodelle angewendet werden.

Unterschiede und wie können diese aneinander angepasst werden, um den gleichen Prozess zu repräsentieren?

2. Welche Modellierungssprache ist für einen konkreten Prozess die geeignetere? Sollte man sich überhaupt auf eine bestimmte Sprache festlegen?

Zur Illustration dieser beiden Fragestellungen werden im Folgenden mehrere Beispiele diskutiert. Dazu werden als Repräsentanten eine deklarative sowie eine imperative Modellierungssprache verwendet. Deklarative Sprachen eignen sich vor allem zur Modellierung von Prozessen, die in ihrer Ausführung viele Freiheitsgrade und mögliche Ausführungspfade besitzen. Imperative Sprachen hingegen sind vor allem sinnvoll für Prozesse, die bei der Ausführung nur wenige Freiheiten bieten und bei denen alle möglichen Ausführungspfade in gewisser Weise schon vormodelliert sind. Als Repräsentant einer deklarativen Modellierungssprache dient hierbei Declare [6, 9], als Repräsentant einer imperativen Modellierungssprache wird Business Process Model and Notation (BPMN)[7] verwendet. An dieser Stelle wären auch andere Modellierungssprachen möglich, da äquivalente Fragestellungen oder Probleme analog bei allen Modellierungssprachen auftreten. Declare und BPMN werden ausgewählt, da diese jeweils der populärste deklarative bzw. imperative Vertreter der Prozessmodellierungssprachen sind. Deshalb werden diese beiden Sprachen in der vorliegenden Arbeit zur Illustration der entwickelten Ansätze und Methoden herangezogen. Als erstes Beispiel sei der im Folgenden beschriebene Prozess P_1 gegeben:

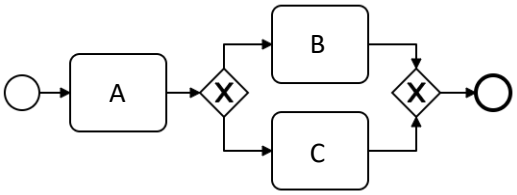
- Es gibt drei verschiedene Aktivitäten: A, B und C.
- Aktivität A muss genau einmal ausgeführt werden.
- Die beiden Aktivitäten B und C dürfen nur ausgeführt werden, falls Aktivität A bereits ausgeführt wurde.
- Die beiden Aktivitäten B und C dürfen nur exklusiv ausgeführt werden (XOR). Eine von beiden Aktivitäten muss ausgeführt werden.
- Jede Aktivität darf maximal einmal ausgeführt werden.

Der dritte Punkt impliziert, dass der Prozess auf jeden Fall mit Aktivität A gestartet werden muss. Da aber alle Aktivitäten auf Grund des letzten Punkts maximal einmal ausgeführt werden dürfen, bleiben für den fortlaufenden Prozess nur noch die beiden Aktivitäten B und C übrig. Der vierte Punkt fordert weiterhin die exklusive Ausführung dieser beiden Aktivitäten. Somit gibt es für P_1 nur zwei gültige Sequenzen:[8] $\langle AB \rangle$ und $\langle AC \rangle$. Abbildung 1.2 zeigt auf der linken Seite

[7] `http://www.omg.org/spec/BPMN/2.0`, abgerufen am: 27. Dezember 2023

[8] In dieser Arbeit wird die in [10] verwendete deutsche Übersetzung *Sequenz* des englischen Begriffs *trace* verwendet. Synonym sind im Deutschen auch die Begriffe *Spur* oder *Ausführungsspur* gebräuchlich.

eine mögliche Modellierung des Prozesses in BPMN. Auf der rechten Seite ist eine äquivalente Darstellung des Prozesses als Declare-Modell dargestellt.[9]



precedence(A, B)	exactly$(A, 1)$
precedence(A, C)	atMost$(B, 1)$
exclusiveChoice(B, C)	atMost$(C, 1)$

Abbildung 1.2: Prozess P_1 dargestellt in BPMN (links) [11] und Declare (rechts)

Zunächst einmal ist es sehr schwierig oder sogar unmöglich, selbst dieses relativ einfache Beispiel per Hand zu verifizieren, d.h. die Gleichheit der beiden Modelle zu zeigen. Vor allem für Domänenexpertinnen und Domänenexperten, die mit einer Modellierungssprache nicht so vertraut sind, ist dieser Vergleich nahezu unmöglich. Damit ergibt sich die Notwendigkeit einer automatisierten Vergleichsmethode. Die tatsächliche Verifikation der Gleichheit der beiden Modelle wird in Kapitel 3.2.4 durchgeführt.

Im Allgemeinen ändern sich Prozesse im Laufe der Zeit, und demzufolge müssen die zugehörigen Prozessmodelle ständig angepasst werden. Im Folgenden soll Prozess P_1 um zwei Sachverhalte erweitert werden:

- Es gibt eine weitere Aktivität D, die jederzeit (allerdings maximal einmal) ausgeführt werden kann.
- Die beiden Aktivitäten B und D dürfen nur exklusiv ausgeführt werden (XOR).

Natürlich muss zur Umsetzung dieser beiden Änderungen sowohl beim BPMN-Modell als auch beim Declare-Modell Aktivität D hinzugefügt werden. Damit die beiden Änderungen im BPMN-Modell integriert werden können, sind einige Anpassungen notwendig. Das Resultat ist in Abbildung 1.3 dargestellt. Um die beiden Änderungen im Declare-Modell anzupassen, sind lediglich die folgenden Anpassungen notwendig: Es muss eine Bedingung atMost$(D, 1)$ hinzugefügt werden, die sicherstellt, dass Aktivität D maximal einmal ausgeführt wird. Eine weitere Bedingung notCoExistence(B, D) garantiert die exklusive Ausführung der beiden Aktivitäten B und D. Abbildung 1.4 zeigt das angepasste Declare-Modell. Man erkennt, dass die Abänderung des Prozesses in Declare deutlich leichter und unkomplizierter durchzuführen ist als in BPMN.

Zusammenfassend verdeutlicht die Vielfalt an Prozessmodellierungssprachen, dass schon vergleichsweise einfache Zusammenhänge in verschiedenen Sprachen zu einer Fülle äquivalenter

[9]Die in diesem Abschnitt verwendeten Modellierungssprachen werden erst in Kapitel 2.1.4 im Detail eingeführt. Leserinnen und Leser, die eine tiefere Verständnisgrundlage wünschen, werden daher gebeten, sich auf das entsprechende Kapitel zu beziehen.

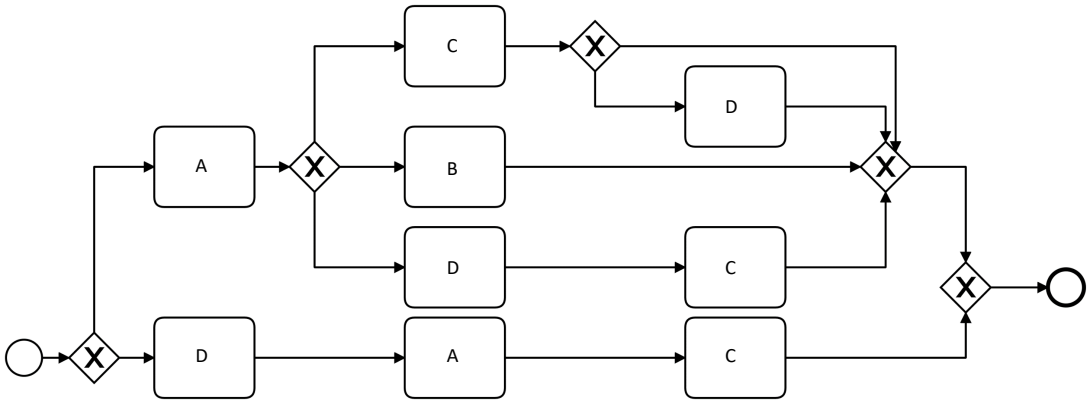


Abbildung 1.3: Angepasstes BPMN-Modell

exactly$(A, 1)$	precedence(A, B)	notCoExistence(B, D) **(NEU)**
atMost$(B, 1)$	precedence(A, C)	exclusiveChoice(B, C)
atMost$(C, 1)$		
atMost$(D, 1)$ **(NEU)**		

Abbildung 1.4: Angepasstes Declare-Modell

Prozessmodelle führen können. Das Fehlen eines eindeutigen Standards verstärkt diese Herausforderung, indem unterschiedliche Abteilungen innerhalb eines Unternehmens verschiedene Modellierungssprachen nutzen können. Die damit einhergehende Problematik stellt grundlegende Fragen hinsichtlich der Gleichheit von Prozessmodellen und der Auswahl geeigneter Modellierungssprachen für spezifische Prozesse. Zudem wird durch Modifikationen oder Erweiterungen von Prozessmodellen deutlich, dass die Anpassung in deklarativen Sprachen im Vergleich zu imperativen Sprachen oft leichter und weniger komplex ist. Selbst geringfügige Veränderungen können bei imperativen Prozessmodellen zu erheblichem Modellierungsaufwand führen. Insbesondere bei der Integration neuer Aktivitäten und der Modifikation von Beziehungen zwischen Aktivitäten zeigt sich die Flexibilität und Effizienz deklarativer Modellierungssprachen. Diese Erkenntnisse legen den Grundstein für die in dieser Arbeit entwickelten Ansätze und Methoden zur Automatisierung von Modellvergleichen. Damit wird das Verständnis für die Herausforderungen der Prozessmodellierung und die Wahl geeigneter Modellierungssprachen vertieft, um den komplexen Anforderungen moderner Unternehmen gerecht zu werden.

1.2 Beitrag der Dissertation

Eines der primären Ziele dieser Abhandlung ist es, ein Konzept zum Vergleich von Prozessmodellen zu entwickeln. Im Prozessmanagement werden grundsätzlich zwei Modellierungsparadigmen

unterschieden: die imperative und die deklarative Prozessmodellierung [3]. Dabei wird die imperative Modellierung bevorzugt für sogenannte Routineprozesse, d.h. Prozesse, deren Ablauf ziemlich genau bekannt ist, verwendet. Im Gegensatz dazu dient die deklarative Prozessmodellierung der Modellierung von flexiblen Prozessen, die eine große Anzahl an möglichen Ausführungspfaden besitzen. Dabei ist es natürlich sehr interessant, deklarative und imperative Prozessmodelle miteinander zu vergleichen, da diese auf den ersten Blick kaum Gemeinsamkeiten aufweisen. Allerdings sollen auch imperative und deklarative Modelle untereinander verglichen werden.

Vor einem Modellvergleich muss zunächst geklärt werden, was unter *gleichen* Prozessmodellen zu verstehen ist, da in der Literatur mehrere verschiedene Gleichheitsbgegriffe verwendet werden. In dieser Arbeit wird die Gleichheit von Prozessmodellen über deren erlaubte Sequenzen definiert: Zwei Prozessmodelle sind gleich, wenn sie die gleichen Sequenzen erlauben [12]. Dabei sollen alle an den Prozessen beteiligten Elemente, z.B. Aktivitäten, ausführende Personen, verwendete Werkzeuge, berücksichtigt werden. Diese Elemente werden in sogenannten *Prozessperspektiven* zusammengefasst [13, 14]. So beschreibt beispielsweise die *Kontrollflussperspektive* die zeitliche Anordnung von Aktivitäten. Da es vom jeweiligen Anwendungsfall abhängig ist, welche Perspektiven der Prozesse verglichen werden sollen, kann die erste Forschungsfrage formuliert werden:

RQ1 Sind zwei Prozessmodelle bezüglich ausgewählter Prozessperspektiven gleich? Die Gleichheit soll unabhängig von der gewählten Modellierungssprache bestimmt werden können.

Zur Beantwortung dieser Frage stellt die vorliegende Arbeit ein Konzept bereit, das basierend auf endlichen deterministischen Automaten einen Modellvergleich durchführt. Dabei werden die zu untersuchenden Prozessmodelle durch verhaltenserhaltende Transformationen auf endliche Automaten abgebildet. Mit Hilfe von zentralen Aussagen der Automatentheorie werden die resultierenden Automaten untersucht und auf Gleichheit überprüft. Daraus können Rückschlüsse auf die ursprünglichen Prozessmodelle gezogen und Aussagen über deren Gleichheit postuliert werden. Der vorgestellte Modellvergleich ist auf alle Prozessmodellierungssprachen anwendbar, die auf endliche Automaten abgebildet werden können. Im Kontext dieser Arbeit werden Declare [6] als Repräsentant einer deklarativen Modellierungssprache und Business Process Model and Notation (BPMN)[10] als Repräsentant einer imperativen Modellierungssprache auf endliche Automaten abgebildet, wodurch der Prozessmodellvergleich ermöglicht wird.

Da im Allgemeinen bei der Ungleichheit von zwei Modellen die bloße Aussage über die Ungleichheit aber nicht genügt, gehen mit **RQ1** auch folgende weitergehende Fragestellungen einher:

- Ist ein Prozessmodell in einem anderen enthalten?

[10] `http://www.omg.org/spec/BPMN/2.0`, abgerufen am: 27. Dezember 2023

- Worin bestehen die Unterschiede zwischen verschiedenen Modellen?
- Worin bestehen die Gemeinsamkeiten von verschiedenen Modellen?
- Wie „ähnlich“ sind ungleiche Modelle?

Die Antworten auf diese Fragen können präzise Aussagen über verschiedene Prozessmodelle liefern. Gerade im Bereich der deklarativen Prozessmodellierung muss die Frage nach gegenseitigem Enthaltensein oftmals für theoretische Fragestellungen beantwortet werden [15, 16]. Bisher wurde diese Fragestellung beantwortet, indem alle möglichen Sequenzen bis zu einer gewissen Länge simuliert wurden und danach überprüft wurde, ob die Menge aller Sequenzen, die von einem Modell akzeptiert wird, eine Teilmenge der Sequenzen, die vom anderen Modell akzeptiert werden, ist. Logischerweise können, da insbesondere deklarative Prozessmodelle oftmals unendlich viele akzeptierte Sequenzen besitzen, nicht alle möglichen Sequenzen berechnet werden. Somit bleibt immer noch eine Restwahrscheinlichkeit, dass bei Übereinstimmen der beiden Modelle auf allen Sequenzen der Länge n eine Sequenz der Länge $> n$ existiert, die nur von einem Modell akzeptiert wird. Folglich würden die beiden Modelle fälschlicherweise als gleich betrachtet werden. Die vorliegende Arbeit liefert unter anderem eine auf Automatentheorie basierende Methodik, die das Überprüfen zweier Prozessmodelle auf gegenseitiges Enthaltensein mit absoluter Gewissheit beantwortet.

Die Fragen in Bezug auf Gemeinsamkeiten und Unterschiede von Prozessmodellen werden in der vorliegenden Arbeit beantwortet, indem, ebenfalls mit Hilfe von Automatentheorie, die sogenannten Differenzsprachen sowie Produktsprachen berechnet werden. Diese beschreiben genau die Unterschiede bzw. Gemeinsamkeiten von Prozessmodellen. Zudem werden Ähnlichkeitsmaße definiert, um die Ausdrucksmächtigkeit und die Ähnlichkeit von verschiedenen Prozessmodellen zu bestimmen.

Da in der Praxis Prozessmodelle immer wieder aktualisiert und geändert werden müssen, um sie an Änderungen an den realen Prozessen anzupassen, stellt sich oftmals die Frage, ob das neue Modell wirklich die gewünschten Eigenschaften besitzt. Insbesondere im Bereich der deklarativen Prozessmodellierung sind Änderungen an Modellen auf Grund des Zusammenspiels mehrerer Regeln oftmals sehr schwierig zu verifizieren. Grundsätzlich sind durch Änderungen an Prozessmodellen vier verschiedene Szenarien möglich:

Szenario 1: Das neue Modell akzeptiert die gleichen Sequenzen wie das ursprüngliche Modell, d.h. das Verhalten hat sich nicht geändert.

Szenario 2: Das neue Modell ist restriktiver, d.h. es akzeptiert nur noch eine Teilmenge der Sequenzen des ursprünglichen Modells.

Szenario 3: Das neue Modell ist permissiver, d.h. es akzeptiert neben den Sequenzen des ursprünglichen Modells noch weitere Sequenzen.

Szenario 4: Das neue Modell akzeptiert sowohl eine echte Teilmenge der Sequenzen des ursprünglichen Modells als auch zusätzliche Sequenzen, die das ursprüngliche Modell nicht akzeptiert hat.

Das erste Szenario bedeutet, dass die beiden Modelle gleich sind. Dies wird bereits in Forschungsfrage **RQ1** beantwortet. Damit müssen also nur noch zwei unterschiedliche Modelle untersucht werden. Die theoretische Antwort, ob gegenseitiges Enthaltensein vorliegt (Szenarien 2 & 3), wurde ebenfalls bereits in **RQ1** gegeben. Hier stellt sich nun noch (ebenso wie für Szenario 4) die Frage, welche Verhaltensänderungen durch die Modellanpassung konkret auftreten. Daraus ergibt sich die zweite Forschungsfrage:

RQ2 Welche Auswirkungen haben Änderungen an einem Prozessmodell?

Die vorliegende Arbeit beantwortet **RQ2**, indem mit Hilfe von Differenzautomaten sowohl das durch Änderungen am Modell verloren gegangene Verhalten als auch das neu hinzugefügte Verhalten berechnet wird.

Wie etliche Studien (z.B. [17, 18, 19]) zeigen, sind deklarative Prozessmodelle für Menschen oftmals schwer zu verstehen und zu interpretieren. Die ihnen zu Grunde liegende Komplexität ist eine der Hauptursachen für ihre relativ geringe Verbreitung und Anwendung in der Praxis. Beispielsweise sind bei der Declare-Repräsentation des Prozesses P_1 (Abbildung 1.2 rechts) im Gegensatz zur imperativen BPMN-Repräsentation (Abbildung 1.2 links) folgende Fragestellungen nicht so einfach manuell zu beantworten:

- Welche Aktivität muss zuerst ausgeführt werden?
- Ist $\langle BA \rangle$ eine gültige Sequenz?

Wenn man dieses Beispiel verallgemeinert, ergibt sich die dritte Forschungsfrage **RQ3**:

RQ3 Wie kann die Verständlichkeit deklarativer Prozessmodelle verbessert werden?

Forschungsfrage **RQ3** ist ein wesentlicher Gegenstand dieser Arbeit. Die Antwort darauf ist die Entwicklung einer szenario-basierten Modellprüfung. Diese Modellprüfung - ebenso basierend auf Automatentheorie - liefert Antworten auf oben gestellte Fragen und unterstützt die Modelliererin und den Modellierer dabei, grundlegende Eigenschaften deklarativer Prozessmodelle zu verstehen und zu verifizieren.

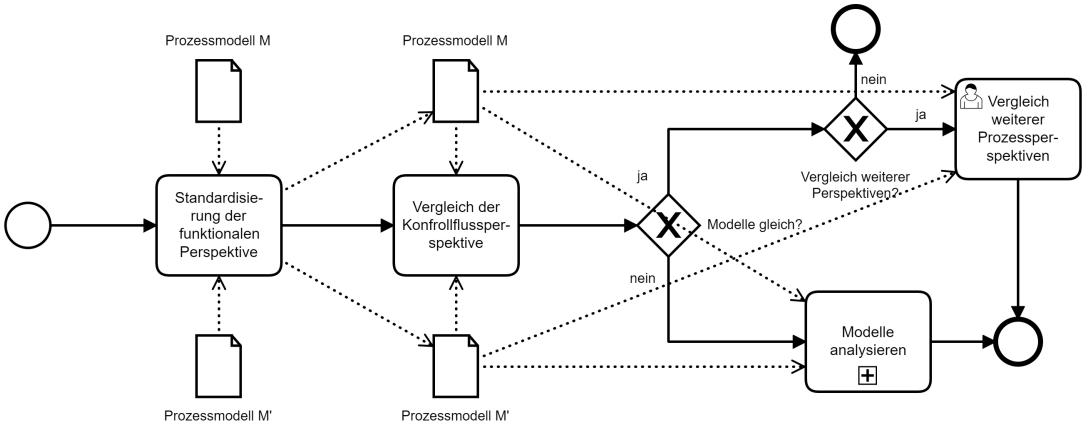


Abbildung 1.5: Gesamtkonzept zum Vergleich von Prozessmodellen

1.3 Gesamtkonzept

In diesem Kapitel wird das Gesamtkonzept der Dissertation skizziert, das aus zwei Teilen besteht. Der erste Teil beantwortet die beiden Forschungsfragen **RQ1** und **RQ2**, die in Kapitel 1.2 formuliert werden. Teil zwei wird sich dann mit Forschungsfrage **RQ3** beschäftigen.

Der erste Teil des Gesamtkonzepts befasst sich mit dem Vergleich von Prozessmodellen. Das vereinfachte Übersichtsschema zum Vergleich von Prozessmodellen ist in Abbildung 1.5 als BPMN-Diagramm dargestellt. Der erste Schritt *Standardisierung der funktionalen Perspektive* (Kapitel 3.1) besteht darin, die beiden zu vergleichenden Modelle M und M' vergleichbar zu machen. Das bedeutet, dass die Namen aller am Prozess beteiligten Elemente, z.B. Aktivitäten oder Personen, übereinstimmen. Dies verhindert beispielsweise, dass die gleiche Aktivität in einem Prozessmodell *Datenanalyse durchführen* und im anderen Prozessmodell *Analyse der Daten* heißt. Dieser Schritt ist für weitere in dieser Arbeit entwickelten Ansätze und Methoden notwendig, da die Prozessmodelle für den Vergleich der Konfrollflussperspektive auf endliche Automaten abgebildet werden und die Aktivitätsnamen die Zustandsübergänge definieren (Kapitel 3.2.1, 3.2.2 und 3.2.3). Demzufolge ist es nur möglich, bei gleich benannten Zustandsüberführungen Vergleichsoperationen durchzuführen. In der Forschung hat sich die Kontrollflussperspektive als primäre Perspektive zur Spezifikation von Abhängigkeiten zwischen Aktivitäten etabliert, weshalb diesem Ansatz vorrangig gefolgt wird. Nachfolgend werden die anderen Perspektiven betrachtet. Jetzt werden die beiden Prozessmodelle im Schritt *Vergleich der Kontrollflussperspektive* mit Hilfe von Automatentheorie bezüglich der Kontrollflussperspektive auf Gleichheit untersucht (Kapitel 3.2.4 und 3.2.6). Falls die Modelle gleich sind, können die Prozessmodelle im Schritt *Vergleich weiterer Prozessperspektiven* noch hinsichtlich weiterer von der Benutzerin oder dem Benutzer selektierter Prozessperspektiven untersucht werden (Kapitel 3.3.1, 3.3.2 und 3.3.3). Falls die beiden

Modelle bezüglich der Kontrollflussperspektive nicht gleich sind, ist eine Untersuchung hinsichtlich weiterer Prozessperspektiven wenig sinnvoll. Daher werden die Prozessmodelle in diesem Fall detaillierter im Hinblick auf die Kontrollflussperspektive analysiert (*Modelle analysieren*). Dabei können beispielsweise, ebenfalls unter Anwendung von Automatentheorie, Gemeinsamkeiten (Kapitel 3.2.8) und Unterschiede (Kapitel 3.2.9) bestimmt werden. Ebenso ist es möglich, die Ähnlichkeit der Prozessmodelle mit Hilfe von Ähnlichkeitsmaßen zu berechnen (Kapitel 5).

Der zweite Teil der vorliegenden Arbeit widmet sich Forschungsfrage **RQ3** (Kapitel 1.2): *Wie kann die Verständlichkeit deklarativer Prozessmodelle verbessert werden?* Dafür wird der Ansatz der Szenario-basierten Modellprüfung entwickelt (Kapitel 6). Dieser besteht aus drei Verhaltensanalysen, die der Prozessmodelliererin und dem Prozessmodellierer bei der Analyse, Modellierung und Überarbeitung deklarativer Prozessmodelle helfen. Auch für diesen Teil liefert die Automatentheorie die theoretische Grundlage.

1.4 Aufbau der Arbeit und Forschungsmethodik

Diese Dissertation ist in ihrem weiteren Verlauf wie folgt strukturiert: Kapitel 2 bietet einen Überblick über die grundlegenden Begriffe und Terminologien, die zur vollständigen Erfassung des Inhalts dieser Arbeit erforderlich sind. In diesem Kapitel werden sowohl die Grundlagen des Geschäftsprozessmanagements als auch die fundamentale Automatentheorie eingeführt, auf der die meisten der in dieser Arbeit entwickelten Methoden basieren. Kapitel 3 widmet sich der Entwicklung von Vergleichsmethoden für Prozessmodelle. Einige der Bausteine dieser Methoden werden in Kapitel 4 verwendet, um die Auswirkungen von Änderungen an Prozessmodellen nachzuvollziehen und zu berechnen. Um die Ähnlichkeit von unterschiedlichen Prozessmodellen quantifizieren zu können, werden in Kapitel 5 spezifische Ähnlichkeitsmaße definiert. Kapitel 6 beschreibt drei verschiedene Szenarien, die das Überarbeiten, Verstehen und Interpretieren von deklarativen Prozessmodellen erleichtern. Nach der detaillierten Darlegung der entwickelten Ansätze in den Kapiteln 3, 4, 5 und 6 erfolgt im Anschluss an das jeweilige Kapitels jeweils eine eingehende Einordnung dieser Konzepte in den Kontext verwandter Arbeiten. Kapitel 7 widmet sich der Umsetzung der Hauptkonzepte dieser Arbeit in zwei Java-Programmen. In Abschnitt 8 werden alle in dieser Arbeit entwickelten Konzepte und Methoden anhand realer Prozessmodelle mit Hilfe der im vorherigen Kapitel beschriebenen Implementierungen im Detail evaluiert. Kapitel 9 gibt eine abschließende Zusammenfassung der Ergebnisse. Außerdem werden potenzielle Ansatzpunkte für zukünftige Forschungsarbeiten erörtert.

In der vorliegenden Arbeit wird weitestgehend die Design-Science-Research-Methode [20] als Grundlage für die Forschung verfolgt. Design Science Research ist eine etablierte Forschungsmethode, die in verschiedenen wissenschaftlichen Disziplinen, insbesondere in der Informatik

und Informationssystemforschung, angewendet wird. Sie zielt darauf ab, Probleme in der realen Welt durch das Entwickeln und Evaluieren von artefaktbezogenen Lösungen zu lösen. Design Science Research geht über die reine Analyse hinaus und betont die Schaffung und Bewertung von Design-Artefakten wie z.B. Software-Systemen, Modellen, Prototypen oder Konzepten.

Dieser Ansatz ermöglicht es, ein praxisorientiertes Problem zu identifizieren, eine entsprechende Lösung zu entwerfen und diese Lösung auf ihre Effektivität und Effizienz zu überprüfen. Durch die Anwendung von Design Science Research wird ein systematischer Rahmen geschaffen, um die Forschungsziele zu erreichen und innovative Erkenntnisse für die Problemstellung zu generieren. Dieser methodische Ansatz erlaubt es, Theorie und Praxis miteinander zu verbinden, um konkrete und anwendbare Ergebnisse zu erzielen, die einen Mehrwert für die jeweilige Domäne bieten.

In der vorliegenden Arbeit werden die folgenden Prinzipien und Leitlinien des Design Science Research angewendet [20]:

1. **Problemrelevanz (engl. *problem relevance*):** Man muss sicherstellen, dass das behandelte Problem oder die Forschungsfrage praktische Relevanz und Bedeutung aufweist und zur Erweiterung des Wissens im jeweiligen Fachgebiet beiträgt.

2. **Design als Artefakt (engl. *design as an artifact*):** Ziel ist es, ein realisierbares Artefakt in Form eines Konstrukts, eines Modells, einer Methode oder einer Instantiierung zu kreieren.

3. **Design als Suchprozess (engl. *design as a search process*):** Der kreative Prozess des Designs ist eine Art Suche, bei der Designerinnen und Designer aktiv nach Lösungen für ein Problem suchen, indem sie Ideen generieren, Alternativen abwägen und iterative Schritte unternehmen, um das optimale Design zu finden. Dieser Prozess beinhaltet oftmals aufwändiges Experimentieren, Testen und Anpassen, damit die bestmögliche Lösung gefunden werden kann.

4. **Forschungsbeiträge (engl. *research contributions*):** Forschungsbeiträge sind die speziellen Erkenntnisse oder Beiträge, die eine Forschungsarbeit zu einem bestehenden Wissensgebiet oder einem bestimmten Problem liefert. Sie tragen dazu bei, Wissenslücken zu schließen oder bestehende Theorien zu erweitern.

5. **Veröffentlichung der Forschung (engl. *communication of research*):** Die Forschungsergebnisse sollten angemessen mit der wissenschaftlichen Gemeinschaft und der Praxis geteilt werden. Dies kann durch wissenschaftliche Veröffentlichungen, Präsentationen oder die Implementierung der Artefakte in realen Anwendungsbereichen erfolgen.

Richtlinie	Kapitel
Problemrelevanz	Kapitel 1.1
Design als Artefakt	Kapitel 3.2, 3.3, 4, 5, 6, 7
Design als Suchprozess	Kapitel 3.4, 4.5, 6.4
Forschungsbeiträge	Kapitel 1.1, 9.1
Veröffentlichung der Forschung	Kapitel 1.5
Wissenschaftliche Sorgfalt	Kapitel 3 bis Kapitel 6
Evaluation des Designs	Kapitel 8

Tabelle 1.1: Einordnung der Arbeit in die Forschungsmethodik Design Science

6. **Wissenschaftliche Sorgfalt (engl. *research rigor*):** Es sind eine strenge methodische Herangehensweise und die systematische Durchführung von Forschungsarbeiten notwendig, um sicherzustellen, dass die Ergebnisse zuverlässig, valide und reproduzierbar sind. Dies umfasst die klare Definition von Forschungszielen, die Auswahl geeigneter Methoden und die sorgfältige Analyse von Daten, um fundierte Schlussfolgerungen zu ziehen.

7. **Evaluation des Designs (engl. *design evaluation*):** Eine umfassende Evaluation der Artefakte sollte durchgeführt werden, um deren Effektivität, Effizienz und Relevanz zu überprüfen. Geeignete Methoden und Kriterien sind hierbei einzusetzen.

Dabei ist zu beachten, dass die spezifischen Anforderungen und Schwerpunkte der Design Science Research je nach Kontext variieren können. Dennoch liegt der Schwerpunkt stets auf dem Designprozess, der Entwicklung von Design-Artefakten und einer sorgfältigen Evaluation, um einen Beitrag sowohl zur Theoriebildung als auch zur praktischen Anwendbarkeit zu leisten.

Die vorliegende Arbeit verfolgte erfolgreich die Design-Research-Methode, was in Tabelle 1.1 deutlich wird, indem sie diese Dissertation in die Kategorie der Design-Science-Research-Methode einordnet. Dieser methodische Ansatz ermöglichte es, systematisch und zielgerichtet an der Lösung komplexer Fragestellungen zu arbeiten, indem sowohl wissenschaftliche Erkenntnisse als auch praktische Anwendungen miteinander verknüpft wurden. Die Anwendung der Design-Science-Research-Methode hat dazu beigetragen, die angestrebten Ziele dieser Arbeit effektiv zu erreichen und innovative Lösungen in ihrem Forschungsbereich zu entwickeln.

1.5 Publikationen des Autors

Einige der in dieser Arbeit vorgestellten Konzepte wurden auf relevanten Konferenzen und in Workshops im Bereich der Informationssysteme für Geschäftsprozesse der wissenschaftlichen Gemeinschaft präsentiert und zur Diskussion gestellt. Die Dissertation baut in Teilen auf bereits veröffentlichten Ansätzen auf, wobei hier jedoch eine tiefergehendere und detailliertere Darstellung

präsentiert wird. Darüber hinaus werden die entwickelten Konzepte in einem breiteren Kontext betrachtet und durch umfangreichere empirische Untersuchungen bekräftigt. Bislang wurden die folgenden thematisch relevanten Arbeiten veröffentlicht:

1. *Upper-Bounded Model Checking for Declarative Process Models [21]*
   N. Schützenmeier, M. Käppel, S. Petter, S. Jablonski, 14th Working Conference on the Practice of Enterprise Modeling, Springer, Riga, 2021.

2. *Automaton-based comparison of Declare process models [22]*
   N. Schützenmeier, M. Käppel, L. Ackermann, S. Jablonski, S. Petter, Software and Systems Modeling 22, 2022.

3. *Scenario-Based Model Checking of Declarative Process Models [18]*
   N. Schützenmeier, M. Käppel, M. Fichtner, S. Jablonski, 25th International Conference on Enterprise Information Systems, SciTePress, Prag, 2023.

4. *Efficient Computation of Behavioral Changes in Declarative Process Models [23]*
   N. Schützenmeier, C. Corea, P. Delfmann, S. Jablonski, 24th International Conference on Business Process Modeling, Development and Support, Springer, Saragossa, 2023.

5. *Comparing the Expressiveness of Imperative and Declarative Process Models [11]*
   N. Schützenmeier, S. Jablonski, M. Käppel, L. Ackermann, 3rd International Workshop on Model-Driven Organizational and Business Agility, Springer, Saragossa, 2023.

Die erstgenannte Publikation beschreibt die erste entwickelte Vergleichsmethode für deklarative Prozessmodelle. Diese wird in der darauf folgenden Publikation um eine effizientere Vergleichsmethode sowie weiterführende und detailliertere Konzepte zur Berechnung von Unterschieden, Gemeinsamkeiten und Ähnlichkeiten deklarativer Prozessmodelle erweitert. Der dritte wissenschaftliche Beitrag untersucht deklarative Prozessmodelle auf essentielle Eigenschaften (sog. „Szenarien“). Publikation vier liefert eine effiziente Berechnung von Verhaltensänderungen bei Änderungen an deklarativen Prozessmodellen. Die letzte der oben aufgeführten Veröffentlichungen erweitert alle bis dahin entwickelten Ansätze für deklarative Prozessmodelle. Diese können nun auch auf imperative Prozessmodelle angewendet werden.

Abschließend bietet dieser Abschnitt einen Blick auf weitere Publikationen, an denen der Autor mitgewirkt hat. Obwohl sie teilweise verschiedene Schwerpunkte aufweisen, tragen sie dennoch entscheidend dazu bei, ein umfassenderes und tiefergehendes Wissen sowie praktische Erfahrungen im Bereich der Informatik und insbesondere des Prozessmanagements zu erlangen. Außerdem sei betont, dass die genannten Publikationen nicht nur eine fundamentale Rolle im Aufbau eines Grundverständnisses im Prozessmanagement spielen, sondern auch für theoretische

Einsichten von großer Bedeutung sind. Diese Arbeiten bieten wertvolle Einblicke, die den Horizont im Bereich der Informatik erweitern und theoretische Grundlagen mit praxisorientierten Erkenntnissen verknüpfen. Durch die Mitwirkung an diesen Publikationen wurde ein solides Verständnis für die Thematik geschaffen. Darüber hinaus wurden auch ein tiefergehendes Wissen sowie praxisnahe Erfahrungen generiert, die einen substantiellen Beitrag zur Weiterentwicklung im Feld des Prozessmanagements leisten.

- *Execution of Multi-perspective Declarative Process Models [24]*
  L. Ackermann, S. Schönig, S. Petter, N. Schützenmeier, S. Jablonski, 26th International Conference on Cooperative Information Systems, Springer, Malta, 2018.

- *Logic Based Look-Ahead for the Execution of Multi-perspective Declarative Processes [25]*
  M. Käppel, N. Schützenmeier, S. Schönig, L. Ackermann, S. Jablonski, 20th International Conference on Business Process Modeling, Development and Support, Springer, Rom, 2019.

- *Detection of Declarative Process Constraints in LTL Formulas [26]*
  N. Schützenmeier, M. Käppel, S. Petter, S. Schönig, S. Jablonski, 15th International Workshop on Enterprise and Organizational Modeling and Simulation, Springer, Rom, 2019.

- *A Comparatative Study for the Selection of Machine Learning Algorithms based on Decriptive Parameters [27]*
  C. Kumar, M. Käppel, N. Schützenmeier, P. Eisenhuth, S. Jablonski, 8th International Conference on Data Science, Technology and Applications, SciTePress, Prag, 2019.

- *Towards a Hybrid Process Modeling Language [8]*
  N. Schützenmeier, S. Jablonski, S. Schönig, 15th International Conference on Research Challenges in Information Science, Springer, Limassol, 2021.

Kapitel 2

Grundlagen

Diese Arbeit widmet sich hauptsächlich einem spezifischen Schwerpunkt innerhalb des Prozessmanagements: dem Vergleich sowie der Analyse unterschiedlicher Prozessmodelle. In diesem Kapitel werden hierfür die notwendigen Definitionen und Konstrukte formuliert. Zunächst werden in Kapitel 2.1 Grundbegriffe des Prozessmanagements sowie die Vertreter der Prozessmodellierungssprachen, auf die sich diese Arbeit fokussieren wird, vorgestellt. Anschließend werden in Kapitel 2.2 Grundlagen der Automatentheorie formuliert, damit diese im Fortgang der Arbeit auf Prozessmodelle angewendet werden können.

2.1 Prozessmanagement

Dieses Kapitel gibt eine Einführung in Grundlagen des Prozessmanagements und insbesondere der Prozessmodellierung. Hierfür wird grundlegende Terminologie definiert (Kapitel 2.1.1) und eine Eingliederung der vorliegenden Arbeit in den Prozesslebenszyklus gegeben (Kapitel 2.1.2). Damit ein Prozess möglichst detailgetreu dargestellt werden kann, folgt anschließend die Einführung der verschiedenen Prozessperspektiven (Kapitel 2.1.3). Danach werden die beiden Modellierungsparadigmen beschrieben: die imperative Prozessmodellierung und die deklarative Prozessmodellierung (Kapitel 2.1.4). Außerdem werden die am weitesten verbreiteten Modellierungssprachen der beiden Paradigmen vorgestellt.

2.1.1 Grundlegende Definitionen

Bei einem **Prozess** handelt es sich nach [28] um eine

> „strukturierte Abfolge von Aktivitäten, die darauf abzielt, bestimmte Inputs in wertvolle Outputs umzuwandeln. Diese Aktivitäten sind miteinander verknüpft und folgen einer festgelegten Reihenfolge, um ein spezifisches Ziel zu erreichen. In Unternehmen

> können Prozesse von unterschiedlichster Art sein, von operativen Abläufen bis hin zu strategischen Entscheidungsprozessen."

Prozessmanagement, andererseits, ist laut [29] ein

> „systematischer Ansatz, um sowohl automatisierte als auch nichtautomatisierte Prozesse zu erfassen, zu gestalten, auszuführen, zu dokumentieren, zu messen, zu überwachen und zu steuern und damit nachhaltig die mit der Unternehmensstrategie abgestimmten Ziele zu erreichen."

Dieser ganzheitliche Ansatz umfasst eine Reihe von Schritten – von der Identifizierung über die Implementierung bis hin zur kontinuierlichen Verbesserung –, um sicherzustellen, dass Prozesse nicht nur funktionieren, sondern auch zur Wertschöpfung beitragen.

Im Folgenden werden grundlegende Begriffe des Prozessmanagements definiert, die für das Verständnis der restlichen Arbeit essentiell sind. Die deutschen Übersetzungen der englischen Begriffe wurden aus [10] übernommen. Zunächst werden die Begriffe *Aktivität* und *Ereignis* definiert:

Definition 1. *Eine **Aktivität** (englisch **activity**) ist ein wohl-definierter Schritt in einem Prozess.*
*Als **Ereignis** (englisch **event**) wird das Auftreten einer Aktivität in einer speziellen Prozessinstanz bezeichnet.*

Eine *Sequenz*[11] spiegelt den zeitlichen Ablauf einer Prozessinstanz, d.h. einer Ausführung eines Prozessmodells, wider:

Definition 2. *Sei $\mathcal{E}$ die Menge aller Ereignisse. Eine **Sequenz** ist eine endliche Anordnung $\sigma = \langle e_1, \ldots, e_n \rangle$ von Ereignissen, sodass alle Ereignisse zur gleichen Prozessinstanz gehören und nach ihrer Ausführungszeit geordnet sind.*
*$n := |\sigma|$ heißt die **Länge von** σ (englisch **length of** σ).*

2.1.2 Prozesslebenszyklus

Ein Modell, das den gesamten Lebenszyklus eines Geschäftsprozesses beschreibt, ist der Prozesslebenszyklus nach Dumas [30]. Dieser ist in Abbildung 2.1 dargestellt. Dieses Modell unterteilt den Lebensweg eines Prozesses in folgende Phasen:

1. **Identifikation:** In dieser Phase werden relevante Prozesse identifiziert und gezielt gestaltet, um die strategischen Ziele der Organisation zu unterstützen.

[11] In dieser Arbeit wird die in [10] verwendete deutsche Übersetzung *Sequenz* des englischen Begriffs *trace* verwendet. Synonym sind im Deutschen auch die Begriffe *Spur* oder *Ausführungsspur* gebräuchlich.

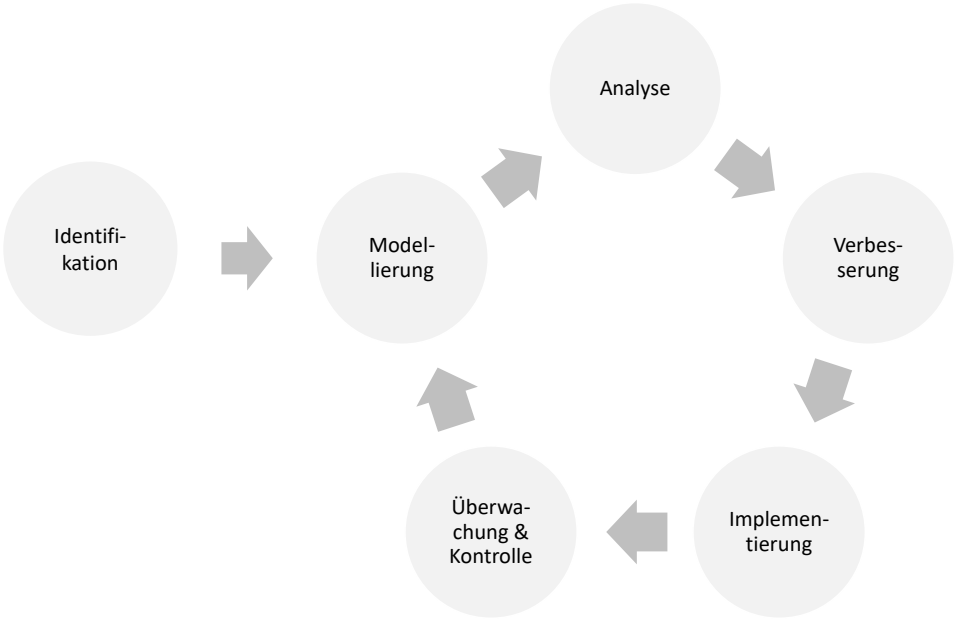


Abbildung 2.1: Prozesslebenszyklus nach Dumas [30]

2. **Modellierung:** Die Prozesse werden visuell dargestellt, um ihre Struktur, Abläufe und Abhängigkeiten zu veranschaulichen.

3. **Analyse:** Hier werden Probleme des Prozesses identifiziert, dokumentiert und, wenn möglich, mithilfe von Prozesskennzahlen quantifiziert.

4. **Verbesserung:** Basierend auf den gesammelten Daten werden Prozesse kontinuierlich verbessert, um Engpässe, Schwachstellen und ineffiziente Bereiche zu eliminieren.

5. **Implementierung:** Die modellierten Prozesse werden technologisch umgesetzt, um die Automatisierung und Steuerung der Abläufe zu ermöglichen.

6. **Überwachung und Kontrolle:** Während der Ausführung werden Kennzahlen überwacht, um die Leistung zu bewerten

Der Prozesslebenszyklus gewährleistet, dass Prozesse nicht nur einmalig optimiert, sondern kontinuierlich an sich ständig verändernde Anforderungen und Marktbedingungen angepasst werden. Dies trägt dazu bei, dass Organisationen agil und wettbewerbsfähig bleiben und gleichzeitig Mehrwert für ihre Kundschaft schaffen.

Die in dieser Arbeit entwickelten Konzepte sind hauptsächlich den beiden Phasen *Modellierung* und *Analyse* zuzuordnen.

2.1.3 Prozessperspektiven

Um einen Prozess möglichst realitätsnah und detailgetreu abbilden zu können, muss eine Vielzahl von Aspekten berücksichtigt werden. Neben der Anordnung von Aktivitäten gehören dazu z.B. auch im Prozess auftretende Personen oder Organisationen und deren Interaktionen. Jeder dieser Aspekte bildet einen sogenannten *orthogonalen Block* [13, 14] und wird als *Prozessperspektive* bezeichnet. Orthogonal bedeutet in diesem Kontext, dass diese Blöcke in folgender Weise unabhängig voneinander sind: Jeder dieser Blöcke kann unabhängig von den anderen Blöcken geändert werden. Außerdem können bei Bedarf Blöcke weggelassen werden, ohne dass dabei Informationen anderer Blöcke verloren gehen. Somit ist es möglich, einen Prozess auf mehreren Ebenen zu beschreiben. In dieser Arbeit werden die folgenden Prozessperspektiven betrachtet [14]:

- **Funktionale Perspektive:** beschreibt auftretende Aktivitäten sowie deren Strukturen (z.B. Subprozesse).
- **Daten- und Datenflussperspektive:** beschreibt Datenobjekte sowie Dokumente. Diese können in Aktivitäten erstellt, bearbeitet oder verwendet werden.
- **Kontrollflussperspektive:** beschreibt den Kontrollfluss, z.B. die kausale und zeitliche Anordnung von Aktivitäten (Sequenzen, parallele Bearbeitung,...).
- **Organisatorische Perspektive:** beschreibt beteiligte Organisationen, z.B. Personen, Rollen oder Systeme. Dabei werden zum einen Beziehungen zwischen Organisationen, zum anderen aber auch Zuordnungen der Aktivitäten zu Organisationen festgehalten.
- **Operationale Perspektive:** beschreibt verwendete Technologien oder Services, z.B. Werkzeuge, Programme, Geräte. Dabei werden zum einen Beziehungen zwischen Technologien, zum anderen aber auch Zuordnungen der Aktivitäten zu Technologien festgehalten.

[31] ergänzt diese fünf Perspektiven um eine weitere:

- **Ortsbezogene Perspektive:** beschreibt die Aufenthaltsorte von Beschäftigten sowie für Aufgaben potentiell vorgegebene Orte.

Ebenso wäre es möglich, eine **Zeitperspektive** zu definieren, die Datums- und Uhrzeitangaben im Modell festlegt [32]. In [33] wurde außerdem die **historische Perspektive** eingeführt. Diese ermöglicht die Verwaltung von Historien in Workflow-Management-Systemen.

Da für die drei zuletzt genannten Prozessperspektiven zu aktuellem Zeitpunkt weder eine geeignete Modellierungs- noch Ausführungsumgebung vorhanden ist, werden diese in dieser Arbeit nicht weiter betrachtet.

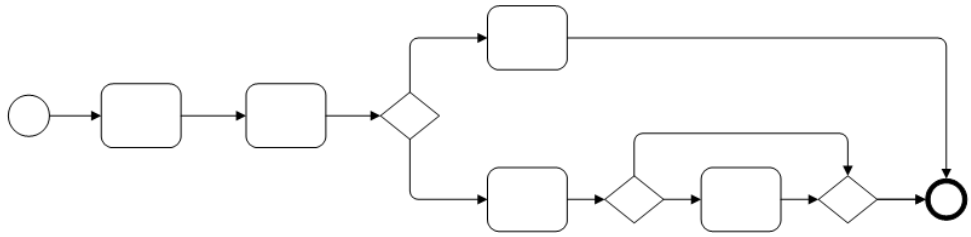


Abbildung 2.2: Imperative Prozessmodellierung

Die funktionale Perspektive besteht aus den im Prozess auftretenden Aktivitäten. Sowohl die Daten- als auch die Kontrollflussperspektive bildet jeweils Flüsse zwischen diesen Aktivitäten. Die organisatorische und operationale Perspektive werden der funktionalen Perspektive zugeordnet. Das bedeutet, dass z.B. Personen oder Werkzeuge, die in externen Modellen (z.B. Organisationsmodellen [14]) verwaltet werden, den Aktivitäten zugeordnet werden. Diese repräsentieren beispielsweise die Ausführende oder den Ausführenden der Aktivität oder das für die Aktivität verwendete Werkzeug.

2.1.4 Prozessmodellierung

Im Allgemeinen werden in der Literatur zwei Arten von Prozessen unterschieden: sogenannte *Routineprozesse* und *flexible Prozesse* [1, 3, 4]. Analog zu diesen beiden Arten existieren zugehörige Prozessmodellierungsparadigmen: die *imperative Prozessmodellierung* und die *deklarative Prozessmodellierung*. Diese werden in den folgenden beiden Unterkapiteln beschrieben. Außerdem werden für jedes Modellierungsparadigma jeweils repräsentierende Beispielmodellierungssprachen vorgestellt.

Routineprozesse und imperative Prozessmodellierung

Bei den sogenannten *Routineprozessen* [3, 4] handelt es sich um Prozesse, deren Ablauf ziemlich genau bekannt und vorgegeben ist. Jede mögliche Ausführung ist in gewisser Weise bereits vormodelliert. Das zugehörige Modellierungsparadigma heißt *imperative Prozessmodellierung*.

Abbildung 2.2 zeigt eine abstrahiertes imperatives Prozessmodell. Dabei repräsentieren die abgerundeten Rechtecke Aktivitäten, die Pfeile dazwischen den Kontrollfluss. Unter der Annahme, dass die Raute bedeutet, dass jeweils genau einer der ausgehenden Pfade durchgeführt werden muss (XOR), existieren ausgehend vom Startpunkt nur drei verschiedene Möglichkeiten, um zu einem Endpunkt zu kommen.

Es gibt einige Vertreter der imperativen Prozessmodellierung. Die bekanntesten und verbreitetsten von ihnen sind die *Ereignisgesteuerte Prozesskette (EPK)* [34], *Yet Another Workflow*

Language (YAWL) [35] und die *Business Process Model and Notation (BPMN)*[12]. In dieser Arbeit wird BPMN als Repräsentant der imperativen Prozessmodellierungssprachen verwendet, da es eine sehr gängige und oft verwendete Modellierungssprache ist, die zudem weitestgehend alle wichtigen Aspekt abdeckt [36]. Abbildung 2.3 zeigt ein BPMN-Modell, anhand dessen nun die wichtigsten Modellierungskonstrukte erklärt werden. Für alle weiteren Konstrukte sei auf die BPMN-Dokumentation verwiesen[12].

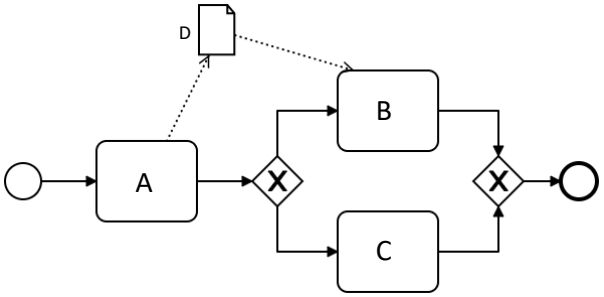


Abbildung 2.3: Beispiel eines BPMN-Modells

Die Hauptkonstrukte von BPMN sind Aktivitäten, Kontrollflüsse, Ereignisse und Gateways. Aktivitäten werden durch Vierecke mit abgerundeten Ecken, die den Aktivitätsnamen enthalten, repräsentiert (im Beispiel gibt es drei Aktivitäten: A, B und C). Konfrollflüsse werden durch Pfeile mit durchgezogener Linie veranschaulicht. Ereignisse werden in Start-, End- und Zwischenereignisse unterteilt. Diese werden durch Kreise dargestellt. Das abgebildete Beispiel zeigt ein Startereignis (ganz links - ohne eingehenden Kontrollfluss) sowie ein Endereignis (ganz rechts - dicke Linie und ohne ausgehenden Kontrollfluss), die jeweils für den Start bzw. das Ende des Prozesses stehen. Gateways (dargestellt durch Rauten) ermöglichen es, Entscheidungen zu modellieren. Dadurch ist es möglich, den Prozessverlauf zu kontrollieren, zu verzweigen oder zusammenzuführen. Das Beispiel zeigt ein sogenanntes öffnendes XOR-Gateway (mit eingehendem Kontrollfluss aus Aktivität A) und ein schließendes XOR-Gateway (mit ausgehendem Kontrollfluss in das Endereignis). Diese werden durch eine Raute, die mit einem X beschriftet ist, dargestellt. XOR bedeutet, dass genau einer der ausgehenden Pfade gewählt werden muss. Im vorliegenden Beispiel gibt es also nur zwei gültige Sequenzen: $\langle AB \rangle$ und $\langle AC \rangle$.

Durch Aktivitäten, Kontrollflüsse, Ereignisse und Gateways werden die funktionale Perspektive und die Kontrollflussperspektive weitestgehend abgedeckt [37]. Das Dokument, das mit einem D bezeichnet ist (Abbildung 2.3), repräsentiert ein Datenobjekt und somit die Datenperspektive. Der aus Aktivität A herausgehende gestrichelte Pfeil sowie der nach B hineingehende gestrichelte Pfeil repräsentieren dabei den Datenfluss (und damit die Datenflussperspektive): In Aktivität A wird Dokument D erstellt und in Aktivität B verwendet. Die operationale Perspektive und die

[12] `http://www.omg.org/spec/BPMN/2.0`, abgerufen am: 27. Dezember 2023

organisatorische Perspektive können in BPMN eingeschränkt durch sogenannte Swimlanes und Pools dargestellt werden [37]. Während Pools und Swimlanes als wichtige Elemente in BPMN-Diagrammen die organisatorische Dimension von Geschäftsprozessen verdeutlichen, existieren sie mehr als Abstraktionen auf der Modellebene und haben begrenzten Einfluss auf die konkrete Ausführungssemantik, die immer durch eine externe Ausführungsumgebung gesteuert werden muss.

Abschließend folgt eine formale Definition eines BPMN-Modells. Dabei werden lediglich Elemente bezüglich der funktionalen Perspektive und der Kontrollflussperspektive verwendet, da diese die für die in dieser Arbeit entwickelten Ansätze ausschlaggebend sind. Zudem ist es auch in der Wissenschaft gängige Praxis, sich bei formalen Untersuchungen auf diese Perspektiven zu beschränken [38].

Definition 3. *Ein **BPMN-Prozessmodell** ist ein Tupel* $P = (F, E, E^s, E^e, A, G, G^p, G^e, T)$*, wobei*

- $F = E \cup A \cup G$ *(sogenannte „Flow-Objects“) eine endliche Menge von Ereignissen* E*, Aktivitäten* A *und Gateways* G *ist,*
- E *eine endliche Menge von Ereignissen ist, die in Start-Ereignisse* E^s *und End-Ereignisse* E^e *unterteilt werden,*
- A *eine endliche Menge von Aktivitäten ist,*
- G *eine endliche Menge von Gateways ist, die in exklusive Gateways* G^e *und parallele Gateways* G^p *unterteilt werden,*
- $T \subseteq F \times F$ *eine endliche Menge von Transitionen ist.*

Flexible Prozesse und deklarative Prozessmodellierung

Eine zweite Kategorie von Prozessen aus Sicht der Anwendung sind die sogenannten *flexiblen Prozesse* [39]. Dies sind Prozesse, während deren Ablauf eine große Anzahl an möglichen Pfaden zur Verfügung steht. Es ist prinzipiell alles erlaubt, was nicht explizit verboten ist. Das zugehörige Modellierungsparadigma heißt *deklarative Prozessmodellierung*. Diese ist in Abbildung 2.4 illustriert. Hier sind alle möglichen Pfade zwischen Start- und Endpunkt, die nicht die rote Zone schneiden oder berühren, erlaubt. Folglich bietet ein deklaratives Prozessmodell im Regelfall viel mehr verschiedene Ausführungspfade als ein imperatives Modell. In Abbildung 2.4 werden exemplarisch vier grüne Pfeile dargestellt, die die Repräsentation von vier zufällig ausgewählten, gültigen Ausführungspfaden veranschaulichen. Die denkbare Abfolge der ausgeführten Aktivitäten „befindet“ sich entlang der gestrichelten Linien.

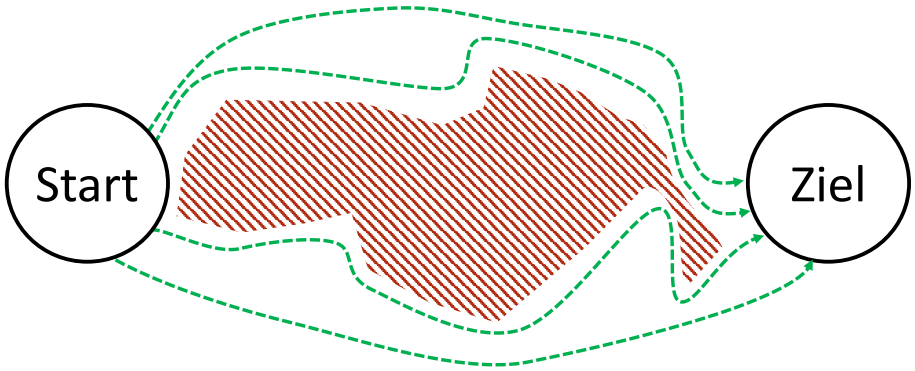


Abbildung 2.4: Deklarative Prozessmodellierung

Operator	Erklärung
$\mathbf{F}\phi$	nächster (Ne**X**t): ϕ gilt am nächsten Zustand.
$\mathbf{G}\phi$	**G**lobal: ϕ gilt auf dem kompletten nachfolgenden Pfad.
$\mathbf{X}\phi$	**F**inally: ϕ gilt irgendwann auf dem nachfolgenden Pfad.
$\psi\mathbf{U}\phi$	**U**ntil: ϕ gilt einschließlich bis zur ersten Position, an der ψ gilt, bis die Position erreicht ist. An dieser Position muss ψ nicht mehr gelten.

Tabelle 2.1: Semantiken der temporalen modalen Operatoren [48]

Vertreter der deklarativen Prozessmodellierung sind z.B. *Declare* [6, 9], *Declarative Process Intermediate Language (DPIL)* [40, 41], *Dynamic Condition Response Graphs (DCR Graphs)* [42], *HiDec* [43], *Guard-Stage-Milestone Lifecycles (GSM)* [44, 45] oder *Multi-Perspective Declare (MP-Declare)* [46], eine multi-perspektivische Erweiterung von Declare.

In dieser Arbeit werden als Repräsentanten der deklarativen Modellierungssprachen Declare sowie die zugehörige multi-perspektivische Erweiterung MP-Declare vorgestellt und als deklarative Modellierungssprache für die Anwendung der entwickelten Methoden und Ansätze verwendet.

Declare ist eine auf der Basis der *linearen temporalen Logik LTL_f über endlichen Sequenzen* [6, 9, 47, 48] definierte deklarative Prozessmodellierungssprache. Dabei werden in LTL_f sogenannte *Constraints* formuliert, die die Einschränkungen für den Prozess festlegen. Eine LTL_f-Formel besteht dabei aus atomaren Aussagen (im Falle von Declare aus den Aktivitäten), den bekannten logischen Operatoren $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ und temporalen modalen Operatoren $\mathbf{F}, \mathbf{G}, \mathbf{X}, \mathbf{U}$ [47, 48]. Die Semantiken der temporalen modalen Operatoren werden in Tabelle 2.1 erklärt. Constraints sind Instanzen von vordefinierten Templates. Declare bietet die Möglichkeit, Bedingungen für eine Aktivität, sogenannte *unäre Constraints*, oder Bedingungen zwischen zwei Aktivitäten, sogenannte *binäre Constraints*, zu formulieren. Declare-Templates sind in der Literatur nicht standardisiert und werden oft in leicht abgeänderter Form (z.B. Umbenennung) verwendet. In dieser Arbeit werden die Templates aus Tabelle 2.2 betrachtet.

Template	**LTL$_f$ Semantiken**
existence(A, n)	$\mathbf{F}(A \wedge \mathbf{X}(\text{existence}(A, n-1))), \text{existence}(A, 1) = \mathbf{F}(A)$
absence(A, n)	$\mathbf{G}(\neg A \vee \mathbf{X}(\text{absence}(A, n-1))), \text{absence}(A, 0) = \mathbf{G}(\neg A)$
init(A)	A
last(A)	$\mathbf{G}(\neg A \rightarrow \mathbf{F}(A))$
respondedExistence(A, B)	$\mathbf{F}(A) \rightarrow \mathbf{F}(B)$
response(A, B)	$\mathbf{G}(A \rightarrow \mathbf{F}(B))$
alternateResponse(A, B)	$\mathbf{G}(A \rightarrow \mathbf{X}(\neg A \mathbf{U} B))$
chainResponse(A, B)	$\mathbf{G}(A \rightarrow \mathbf{X}(B)) \wedge \text{response}(A, B)$
precedence(A, B)	$\mathbf{F}(B) \rightarrow ((\neg B) \mathbf{U} A)$
alternatePrecedence(A, B)	$\text{precedence}(A, B) \wedge \mathbf{G}(B \rightarrow \mathbf{X}(\text{precedence}(A, B))$
chainPrecedence(A, B)	$\text{precedence}(A, B) \wedge \mathbf{G}(\mathbf{X}(B) \rightarrow A)$
succession(A, B)	$\text{response}(A, B) \wedge \text{precedence}(A, B)$
chainSuccession(A, B)	$\mathbf{G}(A \leftrightarrow \mathbf{X}(B))$
alternateSuccession(A, B)	$\text{alternateResponse}(A, B) \wedge \text{alternatePrecedence}(A, B)$
notRespondedExistence(A, B)	$\mathbf{F}(A) \rightarrow \mathbf{F}(B)$
notResponse(A, B)	$\mathbf{G}(A \rightarrow \neg \mathbf{F}(B))$
notChainResponse(A, B)	$\mathbf{G}(A \rightarrow \neg \mathbf{X}(B))$
coExistence(A, B)	$\mathbf{F}(A) \leftrightarrow \mathbf{F}(B)$
notCoExistence(A, B)	$\neg(\mathbf{F}(A) \wedge \mathbf{F}(B))$
choice(A, B)	$\mathbf{F}(A) \vee \mathbf{F}(B)$
exclusiveChoice(A, B)	$(\mathbf{F}(A) \vee \mathbf{F}(B)) \wedge \neg(\mathbf{F}(A) \wedge \mathbf{F}(B))$

Tabelle 2.2: LTL_f-Semantiken der Declare-Templates [22]

Das unäre Template existence$(A, 1)$ bedeutet beispielsweise, dass Aktivität A während der Prozessausführung (mindestens) einmal ausgeführt werden muss. Das binäre Template precedence(A, B) sagt aus, dass Aktivität B nur ausgeführt werden darf, falls zuvor bereits Aktivität A durchgeführt wurde. Für die Erklärung der weiteren Templates sei auf die Literatur [6, 9] verwiesen.

Ein Declare-Constraint kann aufgeteilt werden in den *antecedent* und den *consequent* [49]. Dabei bezeichnet der *antecedent* die Menge an Aktivitäten, die für die Aktivierung des Constraints zuständig sind. Der *consequent* eines Constraints besteht aus den Aktivitäten, die durch den *antecedent* bedingt werden. So ist beispielsweise der *antecedent* von response(A, B) $\{A\}$, der *antecedent* von coExistence(A, B) $\{A, B\}$. Der *consequent* von response(A, B) ist $\{B\}$, der *consequent* von coExistence(A, B) ist ebenfalls $\{A, B\}$. Die Einteilung eines Declare-Constraints in *antecedent* und *consequent* wird benötigt, um in Kapitel 8.3.2 die verwendeten Miner für Declare-Modelle konfigurieren zu können.

Declare besitzt auch eine äquivalente graphische Notation, genannt *ConDec* [6]. Abbildung 2.5

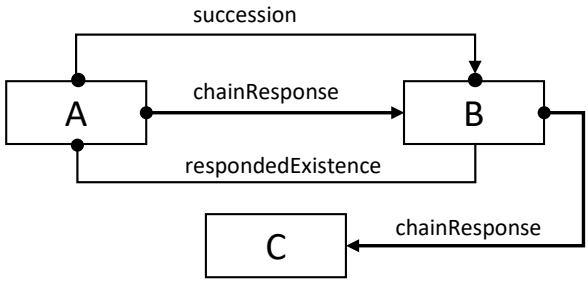


Abbildung 2.5: ConDec-Repräsentation eines Declare-Modells

zeigt die graphische ConDec-Repräsentation eines Declare-Modells mit drei Aktivitäten A, B und C sowie vier Constraints: succession(A, B), chainPrecedence(B, A), respondedExistence(A, B) und chainResponse(B, C). Diese vier Constraints bedeuten:

1. succession(A, B): Falls Aktivität A ausgeführt wird, muss irgendwann danach Aktivität B ausgeführt werden, und falls Aktivität B ausgeführt wird, muss irgendwann zuvor Aktivität A ausgeführt worden sein.

2. chainResponse(A, B): Falls Aktivität A ausgeführt wird, muss direkt danach Aktivität B ausgeführt werden.

3. respondedExistence(A, B): Falls Aktivität A ausgeführt wird, muss irgendwann (zuvor oder danach) Aktivität B ausgeführt werden.

4. chainResponse(B, C): Falls Aktivität B ausgeführt wird, muss direkt danach Aktivität C ausgeführt werden.

Abschließend folgt die formale Definition eines Declare-Modells:

Definition 4. *Ein **Declare-Prozessmodell** ist ein Paar $(\mathcal{A}, \mathcal{T})$, wobei $\mathcal{A}$ eine endliche Menge von Aktivitäten und $\mathcal{T}$ eine endliche Menge von LTL_f-Constraints (d.h. Instanzen der in LTL_f vordefinierten Templates) ist.*

Wie bereits erwähnt, ist Declare eine single-perspektivische Prozessmodellierungssprache und berücksichtigt nur die funktionale Perspektive und die Kontrollflussperspektive. [46] erweiterte Declare um die sogenannten *Aktivierungsbedingung* (englisch *activation condition*) φ_a, *Zielbedingung* (englisch *target condition*) φ_t und *Korrelationsbedingung* (englisch *correlation condition*) φ_c, die in einer beliebigen Logik formuliert werden können. Dadurch erhält man z.B. die Möglichkeit, Prozessvariablen einzuführen und für diese Bedingungen zu formulieren. Zusätzlich kann man Aktivitäten gewisse Personen oder Systeme zuweisen, um die organisatorische und operationale

Template	$\mathbf{LTL}_f$ Semantiken
existence	$\top \rightarrow \mathbf{F}(e(A) \wedge \varphi_a(\vec{x})) \vee \mathbf{O}(e(A) \wedge \varphi_a(\vec{x}))$
respondedExistence	$\mathbf{G}((A \wedge \varphi_a(\vec{x})) \rightarrow (\mathbf{O}(B \wedge \varphi_c(\vec{x}, \vec{y}) \wedge \varphi_t(\vec{y})) \vee \mathbf{F}(B \wedge \varphi_c(\vec{x}, \vec{y}) \wedge \varphi_t(\vec{y}))))$
response	$\mathbf{G}((A \wedge \varphi_a(\vec{x})) \rightarrow \mathbf{F}(B \wedge \varphi_c(\vec{x}, \vec{y}) \wedge \varphi_t(\vec{y})))$
alternateResponse	$\mathbf{G}((A \wedge \varphi_a(\vec{x})) \rightarrow \mathbf{X}(\neg(A \wedge \varphi_a(\vec{x}))\mathbf{U}(B \wedge \varphi_c(\vec{x}, \vec{y}) \wedge \varphi_t(\vec{y})))$
chainResponse	$\mathbf{G}((A \wedge \varphi_a(\vec{x})) \rightarrow \mathbf{X}(B \wedge \varphi_c(\vec{x}, \vec{y}) \wedge \varphi_t(\vec{y})))$
precedence	$\mathbf{G}((B \wedge \varphi_a(\vec{x})) \rightarrow \mathbf{O}(A \wedge \varphi_c(\vec{x}, \vec{y}) \wedge \varphi_t(\vec{y})))$
alternatePrecedence	$\mathbf{G}((B \wedge \varphi_a(\vec{x})) \rightarrow \mathbf{Y}(\neg(B \wedge \varphi_a(\vec{x}))\mathbf{S}(A \wedge \varphi_c(\vec{x}, \vec{y}) \wedge \varphi_t(\vec{y})))$
chainPrecedence	$\mathbf{G}((B \wedge \varphi_a(\vec{x})) \rightarrow \mathbf{Y}(A \wedge \varphi_c(\vec{x}, \vec{y}) \wedge \varphi_t(\vec{y})))$
notRespondedExistence	$\mathbf{G}((A \wedge \varphi_a(\vec{x})) \rightarrow \neg(\mathbf{O}(B \wedge \varphi_c(\vec{x}, \vec{y}) \wedge \varphi_t(\vec{y})) \vee \mathbf{F}(B \wedge \varphi_c(\vec{x}, \vec{y}) \wedge \varphi_t(\vec{y}))))$
notResponse	$\mathbf{G}((A \wedge \varphi_a(\vec{x})) \rightarrow \neg\mathbf{F}(B \wedge \varphi_c(\vec{x}, \vec{y}) \wedge \varphi_t(\vec{y})))$
notChainResponse	$\mathbf{G}((A \wedge \varphi_a(\vec{x})) \rightarrow \neg\mathbf{X}(B \wedge \varphi_c(\vec{x}, \vec{y}) \wedge \varphi_t(\vec{y})))$
notChainPrecedence	$\mathbf{G}((B \wedge \varphi_a(\vec{x})) \rightarrow \neg\mathbf{Y}(A \wedge \varphi_c(\vec{x}, \vec{y}) \wedge \varphi_t(\vec{y})))$

Tabelle 2.3: LTL_f-Semantiken der MP-Declare-Templates [46]

Perspektive abzudecken. Die Daten- und Datenflussperspektive kann durch Anlegen einer Variablen für das jeweilige Datenobjekt (und eventuelle Bedingungen daran) umgesetzt werden. Die Semantiken einiger ausgewählter MP-Declare-Templates sind in Tabelle 2.3 aufgelistet.

2.2 Automatentheorie

Das Ziel der vorliegenden Arbeit ist es, Prozessmodelle zu vergleichen und zu analysieren. Da diese in verschiedenen Modellierungssprachen vorliegen können, ist es erforderlich, eine gemeinsame Repräsentation zu finden, auf deren Basis ein Vergleich möglich ist. In dieser Arbeit wird dafür das Konzept *deterministischer endlicher Automat* [50] verwendet, da dieses sehr gut erforscht ist und zudem alle Grundbausteine für mathematische Operationen (z.B. Produkt von Automaten, Komplement von Automaten) bereitstellt, die für die definierte Aufgabe notwendig sind. Die Grundlagen der Automatentheorie werden in den folgenden Unterkapiteln eingeführt.

2.2.1 Deterministischer endlicher Automat (DEA) und Sprache eines DEAs

Die Definition eines deterministischen endlichen Automats lautet wie folgt:

Definition 5. *Ein **deterministischer endlicher Automat (DEA)** ist ein Quintupel $M = (\Sigma, Q, q_0, \delta, F)$, wobei Σ eine endliche (nicht-leere) Menge von Symbolen, Q eine endliche (nicht-leere) Menge von Zuständen, $q_0 \in Q$ ein Startzustand, $\delta : Q \times \Sigma \rightarrow Q$ eine Zustandsüberführungsfunktion und $F \subseteq Q$ eine Menge von Endzuständen ist.*

Im Kontext des Prozessmanagements werden später Aktivitäten durch Symbole repräsentiert. So entspricht das Ausführen einer Aktivität A in einem Prozess einer mit dem Symbol A

$\delta_1(q_0, A) = q_1$	$\delta_1(q_1, A) = q_1$	$\delta_1(q_2, A) = q_2$
$\delta_1(q_0, B) = q_2$	$\delta_1(q_1, B) = q_1$	$\delta_1(q_2, B) = q_2$
$\delta_2(q_0', A) = q_1'$	$\delta_2(q_1', A) = q_1'$	$\delta_2(q_2', A) = q_2'$
$\delta_2(q_0', B) = q_0'$	$\delta_2(q_1', B) = q_2'$	$\delta_2(q_2', B) = q_2'$

Tabelle 2.4: Zustandsüberführungsfunktionen δ_1 und δ_2 für M_1 und M_2

durchgeführten Zustandsüberführung zwischen zwei Zuständen im korrespondierenden DEA. In diesem Kapitel werden zur Illustration aller Konstrukte und Operationen zwei Beispiel-DEAs verwendet. Diese beiden DEAs M_1 und M_2 sind gegeben durch

$$M_1 = (\{A, B\}, \{q_0, q_1, q_2\}, q_0, \delta_1, \{q_1\}),$$

$$M_2 = (\{A, B\}, \{q_0', q_1', q_2'\}, q_0', \delta_2, \{q_0', q_1'\}).$$

Beide DEAs sind also über dem gleichen Alphabet ($\{A, B\}$) definiert und bestehen aus drei Zuständen. DEA M_1 besitzt einen akzeptierenden Zustand, nämlich q_1, wohingegen DEA M_2 zwei akzeptierende Zustände besitzt (q_0' und q_1'). Die Definitionen der zugehörigen Zustandsüberführungsfunktionen δ_1 und δ_2 sind in Tabelle 2.4 angegeben.

Für endliche Automaten existiert auch eine äquivalente graphische Notation. Die entsprechende Darstellung von M_1 ist in Abbildung 2.6 (links) abgebildet. Dabei entspricht jeder Kreis einem Zustand der Zustandsmenge S. Dieser Kreis beinhaltet den jeweiligen Namen des Zustands. Der Startzustand q_0 ist durch einen eingehenden Pfeil, der selbst aus keinem anderen Zustand kommt, gekennzeichnet. Endzustände (hier q_1) werden durch doppelte Umrandung dargestellt. Die Pfeile zwischen den Zuständen repräsentieren die Zustandsüberführungsfunktion δ_1. So gibt beispielsweise der Pfeil zwischen q_0 und q_1, der mit einem A beschriftet ist, an, dass man von Zustand q_0 durch „Lesen“ von A in den Zustand q_1 übergeht ($\delta_1(q_0, A) = q_1$). Analog ist in Abbildung 2.6 (rechts) der Automat zu M_2 abgebildet.

Bisher gibt die Zustandsüberführungsfunktion nur an, in welchen Zustand man durch das Lesen eines einzelnen Symbols übergeht. Da aber nicht nur einzelne Symbole, sondern auch Aneinanderreihungen von mehreren Symbolen verarbeitet werden sollen, werden die Definitionen von „Wörtern“ sowie der „Länge“ eines Wortes benötigt. Diese Wörter werden an späterer Stelle die Ausführungssequenzen repräsentieren.

Definition 6. *Sei Σ eine endliche (nicht-leere) Menge von Symbolen. Die Menge*

$$\Sigma^* := \{a_1 a_2 \dots a_n \mid n \in \mathbb{N}_0, a_i \in \Sigma\}$$

heißt die ***Menge aller Wörter über*** *Σ.*

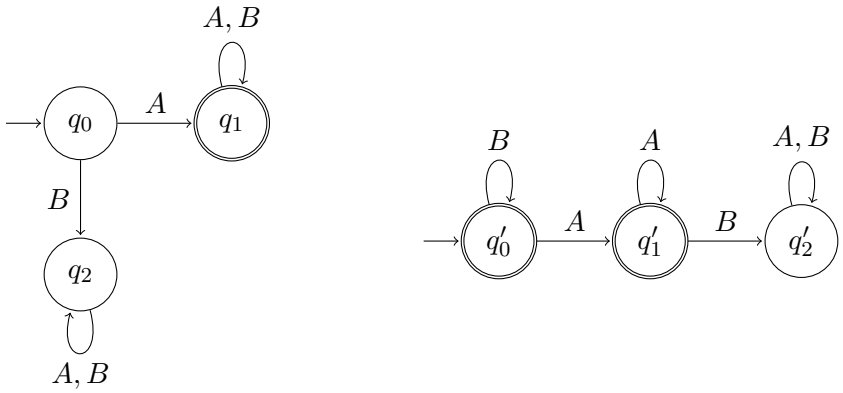


Abbildung 2.6: Graphische Darstellung von M_1 (links) und M_2 (rechts)

*Für jedes Wort $\omega \in \Sigma^*$ ist die **Länge von** ω definiert als*

$$|\omega| := \begin{cases} 0 & \omega = \varepsilon (\varepsilon \text{ bezeichnet das leere Wort}) \\ 1 & \omega \in \Sigma \\ |a| + |b| & \omega = ab \text{ mit } a \in \Sigma \text{ und } b \in \Sigma^*. \end{cases}$$

So gilt beispielsweise $|\varepsilon| = 0, |A| = 1$ und $|ABBAAB| = 6$. Des Weiteren besteht Σ^* für $\Sigma = \{A, B\}$ aus allen Wörtern, die aus den Symbolen A und B gebildet werden können:

$$\Sigma^* = \{A, B\}^* = \{\varepsilon, A, B, AB, BA, AAA, AAB, \dots\}$$

Jetzt kann die Definition der Zustandsüberführungsfunktion auf Wörter erweitert werden:

Definition 7. *Für einen DEA $M = (\Sigma, S, s_0, \delta, F)$ ist die **erweiterte Zustandsüberführungsfunktion** $\hat{\delta}$ definiert durch $\hat{\delta} : S \times \Sigma^* \to S$,*

$$(s, \omega) \mapsto \begin{cases} s & \omega = \varepsilon \\ \delta(s, \omega) & \omega \in \Sigma \\ \hat{\delta}(\delta(s, a), b) & \omega = ab \text{ mit } a \in \Sigma \text{ und } b \in \Sigma^*. \end{cases}$$

Obige Definition bedeutet, dass die erweiterte Zustandsüberführungsfunktion $\hat{\delta}$ sukzessive (von links) die Zustandsüberführungsfunktion δ für die einzelnen Symbole eines Wortes auswertet. Angewandt auf Beispiel M_1 bedeutet dies beispielsweise für das Wort AAB (beginnend in Startzustand q_0):

$$\hat{\delta_1}(q_0, AAB) = \hat{\delta_1}(\delta_1(q_0, A), AB) = \hat{\delta_1}(q_1, AB) = \hat{\delta_1}(\delta_1(q_1, A), B) =$$

$$= \hat{\delta_1}(q_1, B) = \delta_1(q_1, B) = q_1$$

Durch das Lesen des Wortes AAB endet man also im Zustand q_1. Dies lässt sich leicht mit Hilfe der graphischen Darstellung von M_1 (Abbildung 2.6) verifizieren. Im Rest dieser Arbeit wird, um unnötige Verwirrung bei den Notationen zu vermeiden, die erweiterte Zustandsüberführungsfunktion ebenso mit δ (anstatt mit $\hat{\delta}$) bezeichnet.

Um zu überprüfen, ob ein bestimmtes Wort von einem DEA akzeptiert wird, muss folglich nur überprüft werden, ob der Zustand, in dem man sich nach Lesen des Wortes befindet, ein akzeptierender Zustand ist. Ist dies der Fall, wird das Wort akzeptiert. Andernfalls wird es nicht akzeptiert.

Um später Prozessmodelle genauer untersuchen zu können, ist es wichtig, alle Wörter, die von einem DEA akzeptiert werden, zu bestimmen. Diese werden als Sprache des DEAs bezeichnet:

Definition 8. *Sei $M = (\Sigma, Q, q_0, \delta, F)$ ein DEA. Die Menge $\mathcal{L}(M) := \{\omega \in \Sigma^* \mid \delta(q_0, \omega) \in F\} \subseteq \Sigma^*$ heißt die **Sprache von** M.*

Die Sprache eines DEAs $M = (\Sigma, Q, q_0, \delta, F)$ besteht also aus allen Wörtern $\omega \in \Sigma^*$, die (ausgehend vom Startzustand q_0) in einem akzeptierenden Zustand enden.

Um im Beispiel-DEA M_1 in den einzigen akzeptierenden Zustand q_1 zu kommen, muss als erstes Symbol ein A gelesen werden, da man mit einem B im nicht-akzeptierenden Zustand q_2 landet. Dieser kann auf Grund der reflexiven Transition mit beiden Symbolen A und B nicht mehr verlassen werden. In Zustand q_1 darf sowohl A als auch B gelesen werden. Somit besteht die Sprache von M_1 aus allen Wörtern $\omega \in \{A, B\}^*$, die mit dem Symbol A beginnen [21]:

$$\mathcal{L}(M_1) = \{A\omega \mid \omega \in \{A, B\}^*\}$$

Eine analoge Betrachtung ergibt, dass $\mathcal{L}(M_2)$ aus allen Wörtern $\omega \in \{A, B\}^*$ besteht, die entweder kein A beinhalten oder in denen nach einem A kein B mehr folgt:

$$\mathcal{L}(M_2) = \{\varepsilon, A, B, BA, BB, AA, AAA, BAA, BBA, \dots\}$$

2.2.2 Reguläre Ausdrücke

In Kapitel 2.2.1 werden Wörter und Sprachen von endlich deterministischen Automaten eingeführt. Diese Wörter bzw. Sprachen werden später die Ausführungssequenzen von Prozessmodellen repräsentieren. Neben der Darstellung von Sprachen als Automaten gibt es eine weitere äquivalente algebraische Darstellung, sogenannte *reguläre Ausdrücke*. Mit Hilfe dieser theoretischen Konstrukte können essentielle Aussagen über Sprachen bzw. Prozessmodelle getroffen werden, die nur mit der

Automatendarstellung oftmals sehr schwer oder gar nicht zu finden sind. Im Folgenden werden Grundlagen von regulären Ausdrücken beschrieben.

Reguläre Ausdrücke basieren auf drei verschiedenen Operationen: der Alternative (mathematisches „oder“), der Verkettung (entspricht der Konkatenation von Wörtern) sowie der Wiederholung (von Zeichenketten). Dies wird wie folgt formal definiert [50]:

Definition 9. *Sei Σ eine endliche Menge von Symbolen.* ***Reguläre Ausdrücke*** *über Σ sind folgendermaßen definiert:*

1. *Die leere Menge $\emptyset$ ist ein regulärer Ausdruck.*
2. *Jedes Symbol $a \in \Sigma$ ist ein regulärer Ausdruck.*
3. *Für reguläre Ausdrücke a_1, a_2 sind auch $(a_1|a_2)$ (Alternative), (a_1a_2) (Verkettung) und $(a_1)^*$ („Kleenesche Hülle“, Wiederholung) reguläre Ausdrücke.*

Für die Alternative wird in der Literatur oftmals das Symbol + statt | verwendet. Man schreibt dann $(a_1 + a_2)$ für $(a_1|a_2)$. Ebenso ist es gebräuchlich, $\circ$ als Operatorsymbol für die Verkettung zu verwenden. Man schreibt dann $(a_1 \circ a_2)$ statt (a_1a_2).

Für den Fall, dass man eine Rangfolge der Operatoren angibt, kann man auf etliche Klammern verzichten. In dieser Arbeit wird die gängige Rangreihenfolge Kleene-Stern vor Verkettung vor Alternative verwendet. Somit lässt sich beispielsweise der reguläre Ausdruck $(((a_1a_2)|a_3)^*)$ äquivalent als $(a_1a_2|a_3)^*$ formulieren.

Ein regulärer Ausdruck beschreibt analog zu den endlich deterministischen Automaten (Kapitel 2.2.1) eine Sprache. Die Sprache $\mathcal{L}(a)$ eines regulären Ausdrucks a über einem Alphabet Σ wird analog zur Definition dessen Syntax definiert:

1. $\mathcal{L}(\emptyset) = \emptyset$ für die leere Menge $\emptyset$.
2. $\mathcal{L}(a) = \{a\}$ für jedes Symbol $a \in \Sigma$.
3. Für zwei reguläre Ausdrücke a_1 und a_2 gilt:
   - $\mathcal{L}(a_1|a_2) = \mathcal{L}(a_1) \cup \mathcal{L}(a_2)$
   - $\mathcal{L}(a_1a_2) = \{\alpha\beta \mid \alpha \in \mathcal{L}(a_1) \wedge \beta \in \mathcal{L}(a_2)\}$
   - $\mathcal{L}(a_1^*) = \{\alpha_1 \dots \alpha_n \mid n \in \mathbb{N}_0, \alpha_1, \dots, \alpha_n \in \mathcal{L}(a_1)\}$

Reguläre Ausdrücke sind gleich mächtig zu endlich deterministischen Automaten [50]. Das bedeutet, dass zu jedem DEA ein äquivalenter regulärer Ausdruck existiert und umgekehrt. In dieser Arbeit ist lediglich die Konstruktion eines regulären Ausdrucks aus einem DEA relevant, um Prozessmodelle auf theoretischer Basis genauer beschreiben zu können. Dieser Ausdruck kann mit

Hilfe von Standardtechniken aus der Automatentheorie [50] berechnet werden. Reguläre Ausdrücke sind im Allgemeinen nicht eindeutig und können oftmals auch sehr groß und unübersichtlich werden. Das Berechnen von regulären Ausdrücken aus minimalen DEAs liefert dennoch in der Regel nachvollziehbare und aussagekräftige Ergebnisse [50].

Beispielhaft werden die regulären Ausdrücke zu den Automaten M_1 und M_2 über dem Alphabet $\Sigma = \{A, B\}$ aus Kapitel 2.2.1 betrachtet. Ein äquivalenter regulärer Ausdruck zu M_1 ist $a_1 = A(A|B)^*$. $\mathcal{L}(a_1)$ besteht aus allen Wörtern, die mit A beginnen, analog zur Sprache von M_1. Ein entsprechender regulärer Ausdruck zu M_2 wäre $a_2 = B^*A^*$.

2.2.3 Nichtdeterministischer endlicher Automat (NEA)

In der theoretischen Informatik gibt es neben den deterministischen endlichen Automaten (Kapitel 2.2.1) noch eine weitere Art von Automaten: die *nichtdeterministischen endlichen Automaten (NEA)*. Diese werden benötigt, um die Konkatenation von DEAs berechnen zu können. Die formale Definition eines NEAs lautet:

Definition 10. *Ein **nichtdeterministischer endlicher Automat (NEA)** ist ein Quintupel $M = (\Sigma, Q, q_0, \delta, F)$, wobei Σ eine endliche (nicht-leere) Menge von Symbolen, Q eine endliche (nicht-leere) Menge von Zuständen, $q_0 \in Q$ ein Startzustand, $\delta : Q \times \{\Sigma \cup \{\varepsilon\}\} \to \mathcal{P}(Q)$ eine Zustandsüberführungsfunktion und $F \subseteq Q$ eine Menge von Endzuständen ist.*

Ein NEA unterscheidet sich von einem DEA also nur in der Definition der Zustandsüberführungsfunktion. Hier wird beim NEA die Definition um zwei Eigenschaften erweitert:

1. Als Symbol für den Zustandsübergang ist hier auch das leere Wort ε erlaubt. Somit kann es Übergänge geben, in denen einfach nur „weitergesprungen“ wird, ohne dass dabei ein richtiges Symbol gelesen wird. Dies ist die für diese Arbeit wichtige Eigenschaft von NEAs. Sie wird später benötigt, um DEAs miteinander zu konkatenieren.

2. Die Zustandsüberführungsfunktion δ liefert nicht nur einen Zustand (wie beim DEA) zurück, sondern ein Element der Potenzmenge der Zustände (und somit unter Umständen mehrere Zustände). Dies bedeutet, dass aus einem Zustand die Zustandsüberführung durch ein Symbol nicht deterministisch ist. Diese Eigenschaft ist für die vorliegende Arbeit allerdings nicht von Relevanz.

Abbildung 2.7 zeigt einen NEA M mit drei Zuständen q_0, q_1 und q_2. In Zustand q_0 ist es durch Lesen des Zeichens A entweder möglich, in Zustand q_0 zu bleiben oder aber in Zustand q_1 zu springen. Außerdem gibt es einen ε-Übergang von Zustand q_1 nach Zustand q_2. Dieser NEA akzeptiert beispielsweise die Wörter A, AA, BA, BAA, BBA.

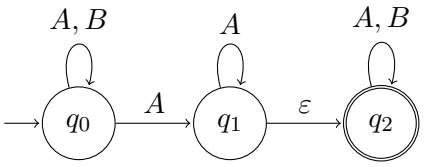


Abbildung 2.7: Nichtdeterministischer Automat M

Aus der Definition eines DEAs (Kapitel 2.2.1) folgt direkt, dass jeder DEA auch ein NEA ist. Etwas erstaunlicher ist allerdings die Tatsache, dass jeder NEA in einen äquivalenten DEA umgewandelt werden kann:

Satz 1. *Sei $M = (\Sigma, Q, q_0, \delta, F)$ ein NEA. Dann gibt es einen DEA $M' = (\Sigma, Q, q_0, \delta, F)$, sodass $\mathcal{L}(M) = \mathcal{L}(M')$.*

Jeder NEA kann via Potenzmengenkonstruktion in einen äquivalenten DEA umgewandelt werden [50]. Diese Konstruktion wird im Laufe dieser Arbeit verwendet, um aus den durch die Konkatenation von DEAs erhaltenen NEAs wieder einen DEA zu bekommen. Abbildung 2.8 zeigt einen zu in Abbildung 2.7 abgebildeten NEA M äquivalenten DEA M'.

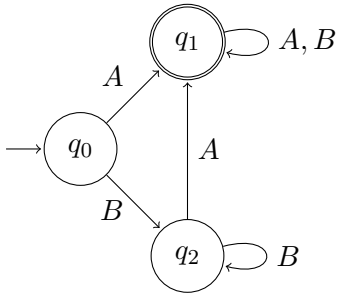


Abbildung 2.8: Zu M äquivalenter DEA M'

2.2.4 Konkatenation deterministischer endlicher Automaten

Für spätere Transformationen von Prozessmodellen in DEAs wird die Kombination von verschiedenen Sprachen und insbesondere deren repräsentierenden Automaten wichtig sein. Dafür können zwei verschiedene DEAs konkateniert werden:

Definition 11. *Seien $M_1 = (\Sigma, Q_1, q_{0_1}, \delta_1, F_1)$ und $M_2 = (\Sigma, Q_2, q_{0_2}, \delta_2, F_2)$ DEAs über der gleichen Menge von Symbolen Σ. Der NEA*

$$M_1 \circ M_2 := (\Sigma, Q_1 \cup Q_2, q_{0_1}, \delta, F_2)$$

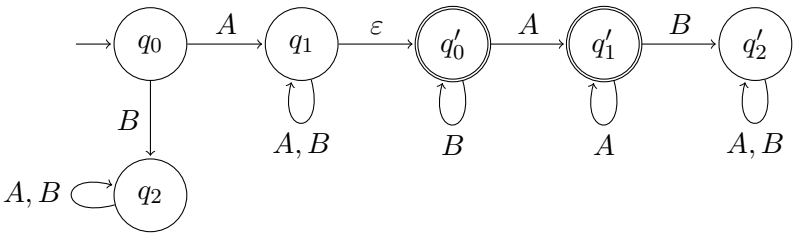


Abbildung 2.9: Konkatenation von M_1 und M_2

heißt die Konkatenation von M_1 und M_2*, wobei*

$$\delta(q,\omega) := \begin{cases} \delta_1(q,\omega) & q \in Q_1 \\ \delta_2(q,\omega) & q \in Q_2 \\ q_{0_2} & q \in F_1 \wedge \omega = \varepsilon. \end{cases}$$

Die Konkatenation von zwei DEAs ist also nichts anderes als das „Hintereinanderschalten“ der beiden Automaten M_1 und M_2 durch Einfügen von ε-Übergängen von allen akzeptierenden Zuständen von M_1 in den Startzustand von M_2. Somit kann nach Lesen eines Wortes in der Sprache von M_1 in den Automaten M_2 „gesprungen“ werden. Das bedeutet, dass die Sprache der Konkatenation zweier Automaten M_1 und M_2 aus sämtlichen Wortkombinationen der beiden Automaten besteht:

$$\mathcal{L}(M_1 \circ M_2) = \{\omega_1\omega_2 \mid \omega_1 \in \mathcal{L}(M_1), \omega_2 \in \mathcal{L}(M_2)\}$$

In Abbildung 2.9 wird exemplarisch die Konkatenation der beiden Automaten M_1 und M_2 aus Abbildung 2.6 veranschaulicht. Der linke Abschnitt, bestehend aus den Zuständen q_0, q_1 und q_2, bildet den ersten DEA M_1. Durch Verwendung eines ε-Übergangs zwischen q_1 und q_0' wird die Konkatenation visualisiert und somit eine Verbindung zwischen M_1 und M_2 hergestellt. Die drei Zustände q_0', q_1' und q_2' repräsentieren dabei den zweiten DEA M_2.

2.2.5 Komplementautomat

Bei der Untersuchung und Überarbeitung von Prozessmodellen wird es wichtig sein, sogenanntes „ungewolltes Verhalten“ zu identifizieren. Hierbei handelt es sich um Sequenzen, die im Prozessmodell irrtümlich erlaubt sind. Diese Aufgabe wird später ebenfalls mit Hilfe der Automatentheorie gelöst, indem die sogenannte *Komplementsprache* und der zugehörige *Komplementautomat* eines DEAs verwendet werden:

Definition 12. *Sei $M = (\Sigma, Q, q_0, \delta, F)$ ein DEA. Dann heißt*

$$\mathcal{L}(M)^C := \Sigma^* \setminus \mathcal{L}(M)$$

*die **Komplementsprache** oder das **Komplement von** $\mathcal{L}(M)$.*
*Weiter heißt $M^C := (\Sigma, Q, q_0, \delta, Q \setminus F)$ der **Komplementautomat von** M.*

Zu einem gegebenen DEA erhält man den zugehörigen Komplementautomaten also durch Vertauschen aller nicht-akzeptierenden und akzeptierenden Zustände. Somit akzeptiert der Komplementautomat genau alle Wörter, die der ursprüngliche DEA nicht akzeptiert hat, kurz:

$$\mathcal{L}(M^C) = \mathcal{L}(M)^C$$

Für den Beispiel-DEA M_1 (Abbildung 2.6 links) bedeutet dies, dass $\mathcal{L}(M_1)^C$ aus allen Wörtern, die nicht mit A beginnen, und dem leeren Wort ε, besteht:

$$\mathcal{L}(M_1)^C = \{\varepsilon, B, BA, BB, BBA, \dots\}$$

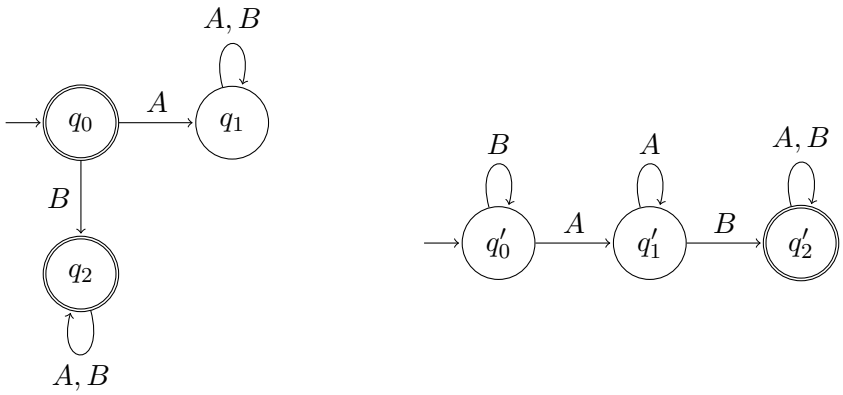


Abbildung 2.10: Graphische Darstellung von M_1^C (links) und M_2^C (rechts)

Der DEA für den Komplementautomaten M_1^C ist in Abbildung 2.10 (links) dargestellt. Analog zeigt Abbildung 2.10 (rechts) den Komplementautomaten M_2^C. Dieser Automat akzeptiert alle Wörter, bei denen mindestens ein B auf ein A folgt, da der einzige akzeptierende Zustand q_2' nur erreicht werden kann, wenn mindestens ein A gelesen wird (Übergang von q_0' nach q_1') und danach irgendwann ein B gelesen wird (Übergang von q_1' nach q_2').

2.2.6 Produktautomat

Um Gemeinsamkeiten von zwei DEAs (und somit später von Prozessmodellen) herauszuarbeiten, ist es hilfreich, die sogenannte Produktsprache zu untersuchen. Mit deren Hilfe werden später Gemeinsamkeiten von Prozessmodellen ermittelt. Die Produktsprache entspricht genau dem Schnitt der Sprachen der beiden betrachteten Automaten. Diese Sprache kann mit Hilfe des Produktautomats berechnet werden:

Definition 13. *Seien* $M_1 = (\Sigma, Q_1, q_{0_1}, \delta_1, F_1)$ *und* $M_2 = (\Sigma, Q_2, q_{0_2}, \delta_2, F_2)$ *DEAs über der gleichen Menge von Symbolen* Σ*. Der* ***Produktautomat*** $M = M_1 \times M_2$ *ist definiert als* $M := (\Sigma, Q_M, q_{0_M}, \delta_M, F_M)$*, wobei* $Q_M = Q_1 \times Q_2$*,* $q_{0_M} = (q_{0_1}, q_{0_2})$*,* $\delta_M : S \times \Sigma \to Q, ((q_1, q_2), a) \mapsto (\delta_1(q_1, a), \delta_2(q_2, a))$*, und* $F_M = F_1 \times F_2$*.*

Aus der Definition des Produktautomatens $M = M_1 \times M_2$ folgt, dass die Sprache des Produktautomaten genau der Schnitt der Sprachen der beiden Automaten M_1 und M_2 ist:

$$\mathcal{L}(M) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$$

Des Weiteren folgt aus der Konstruktion des Produktautomaten M, dass dieser $|S_1| \cdot |S_2|$ Zustände besitzt. Abbildung 2.11 zeigt den Produktautomat $M = M_1 \times M_2$ der Beispielautomaten M_1 (Abbildung 2.6 links) und M_2 (Abbildung 2.6 rechts). Da kein Übergang in $q_1 q_0^{'}$ führt, ist $q_1 q_1^{'}$ der einzige akzeptierende Zustand von M, der erreicht werden kann. Diesen erreicht man aus dem Startzustand $q_0 q_0^{'}$ durch das Symbol A. Da man durch ein A in diesem Zustand verbleibt und durch ein B in Zustand $q_1 q_1^{'}$ (den man nicht mehr verlassen kann) landet, erkennt M nur noch Wörter, die ausschließlich aus dem Symbol A bestehen:

$$\mathcal{L}(M) = \{A\omega \mid \omega \in \{A\}^*\}$$

2.2.7 Minimalisierung

Bei näherer Betrachtung von Abbildung 2.11 fällt auf, dass der Produktautomat M noch sogenannte „unerreichbare Zustände“ beinhaltet. Ein unerreichbarer Zustand ist ein Zustand $\tilde{q}$, den man durch kein Wort (beginnend im Startzustand $q_0 q_0^{'}$) erreichen kann, d.h. $\nexists\, \omega \in \Sigma^* : \delta(q_0, \omega) = \tilde{q}$. Beispielsweise besitzt der Zustand $q_1 q_0^{'}$ keine eingehende Transition und ist somit ein unerreichbarer Zustand. Da derartige Zustände bei einer möglichen Implementierung natürlich unnötig Speicher beanspruchen, ist es wünschenswert, einen DEA so klein wie möglich zu halten. Dafür werden diese unerreichbaren Zustände entfernt. Eine mögliche Vorgehensweise zur Berechnung

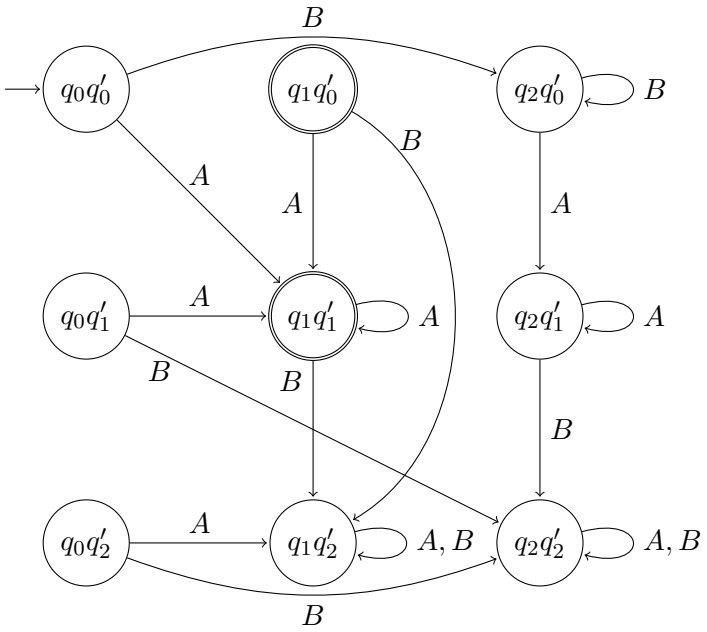


Abbildung 2.11: Produktautomat $M = M_1 \times M_2$

von unerreichbaren Zuständen besteht darin, den DEA als gerichteten Graphen zu betrachten und darauf eine Breitensuche durchzuführen [50]. Alle Zustände, die bei dieser Breitensuche nicht erreicht werden, sind somit unerreichbare Zustände. Die berechneten unerreichbaren Zustände können einfach entfernt werden. Ebenso werden alle Zustandsüberführungen, die aus einem unerreichbaren Zustand herausgehen oder in einen unerreichbaren Zustand führen, gelöscht. Dieser Algorithmus besitzt eine Laufzeit von $\mathcal{O}(|Q| + n)$, wobei n die Anzahl der Zustandsübergänge von M ist. Das Berechnen und Entfernen unerreichbarer Zustände erfolgt also in linearer Laufzeit und ist demnach sehr praktikabel.

Es gibt noch eine weitere Möglichkeit, einen DEA zu vereinfachen. So kann ein erreichbarer Zustand äquivalent zu einem anderen erreichbaren Zustand sein. Zum Beispiel landet man durch Lesen eines Bs in M (Abbildung 2.11) vom Startzustand aus im Zustand $q_2q_0^{'}$. Von $q_2q_0^{'}$ ausgehend können allerdings nur noch die Zustände $q_2q_1^{'}$ sowie $q_2q_2^{'}$ erreicht werden. Da diese aber alle nicht-akzeptierende Zustände sind, sind sie folglich nicht notwendig. Ein Wort, das mit B anfängt, wird von M niemals akzeptiert. Somit können diese drei Zustände zu einem Zustand (inklusive einem mit A, B beschrifteten reflexiven Übergang) zusammengefasst werden. Zur Berechnung dieser äquivalenten Zustände findet man in der Literatur zahlreiche Algorithmen. In dieser Arbeit wird der *Hopcroft Algorithmus* [51] verwendet, der zum aktuellen Stand der laufzeittechnisch effizienteste ist. Dieser hat eine asymptotische Laufzeit von $\mathcal{O}(|\Sigma||Q| \log(|Q|))$. Jetzt kann der sogenannte „Minimalautomat“ definiert werden, der keine unerreichbaren und

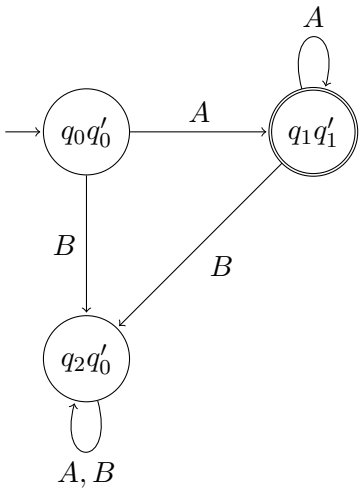


Abbildung 2.12: Minimalautomat M_{min} zu $M = M_1 \times M_2$

keine äquivalenten Zustände mehr besitzt:

Satz 2. *Zu jedem DEA $M = (\Sigma, Q, q_0, \delta, F)$ existiert ein minimaler Automat M_{min} mit $\mathcal{L}(M) = \mathcal{L}(M_{min})$. M_{min} ist bis auf Benennung der Zustände eindeutig und heißt **Minimalautomat**.*

Dieser Minimalautomat kann berechnet werden, indem man zunächst alle unerreichbaren Zustände entfernt und anschließend den Hopcroft Algorithmus anwendet. Dieses Vorgehen wird in dieser Arbeit zur Berechnung des Minimalautomaten Verwendung finden.

Abbildung 2.12 zeigt den äquivalenten Minimalautomaten M_{min} zu $M = M_1 \times M_2$ (Abbildung 2.11). Man sieht, dass sich die Anzahl der Zustände von neun auf drei reduziert hat.

2.2.8 Differenzautomat

Da man oft nicht nur an den Gemeinsamkeiten von Automaten, sondern auch an deren Unterschieden interessiert ist, wird im Folgenden der sogenannte *Differenzautomat* konstruiert.

Definition 14. *Seien M_1 und M_2 DEAs über der gleichen Menge von Symbolen Σ. Dann heißt*

$$M_1 \setminus M_2 := M_1 \times M_2^C$$

*der **Differenzautomat von** M_1 **und** M_2.*

Aus der Konstruktion ergibt sich unmittelbar, dass $\mathcal{L}(M_1 \setminus M_2) = \mathcal{L}(M_1) \setminus \mathcal{L}(M_2)$. Dies impliziert, dass die Sprache des Differenzautomaten $\mathcal{L}(M_1 \setminus M_2)$ aus allen Wörtern besteht, die von M_1 akzeptiert werden, jedoch nicht von M_2.

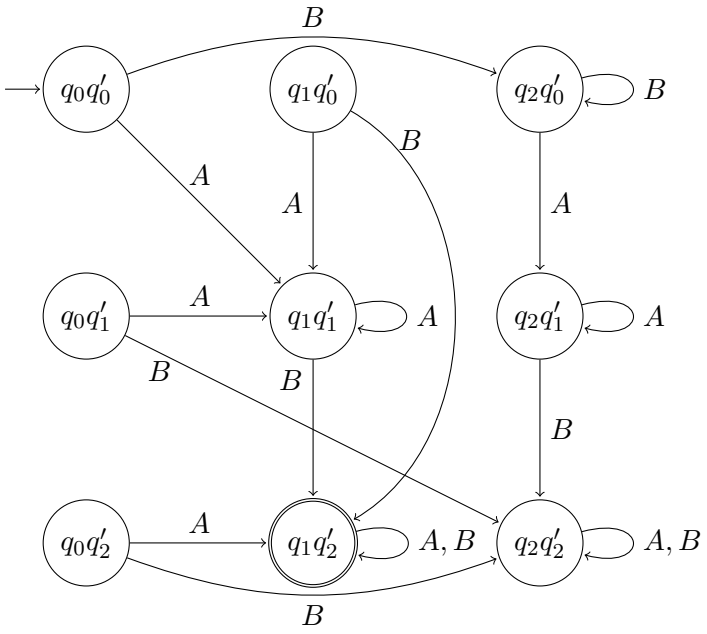


Abbildung 2.13: Differenzautomat $M_1 \setminus M_2 = M_1 \times M_2^C$

Man beachte, dass es zu zwei Automaten M_1 und M_2 jeweils zwei (im Allgemeinen) verschiedene Differenzautomaten gibt, nämlich einen Automaten für die Differenzsprache $\mathcal{L}(M_1) \setminus \mathcal{L}(M_2)$ sowie einen Automaten für die Differenzsprache $\mathcal{L}(M_2) \setminus \mathcal{L}(M_1)$. Abbildung 2.13 zeigt den Differenzautomaten $M_1 \setminus M_2$ für die beiden Beispielautomaten M_1 und M_2 aus Abbildung 2.6. Der Differenzautomat $M_2 \setminus M_1$ unterscheidet sich lediglich in den akzeptierenden Zuständen vom Produktautomaten. Ansonsten sind beide topologisch identisch, d.h. sie besitzen die gleiche Anzahl an Zuständen und die gleichen Zustandsübergänge. Daher wird $M_2 \setminus M_1$ nicht graphisch abgebildet. $M_2 \setminus M_1$ besitzt die folgenden vier akzeptierende Zustände: $q_0q_0^{'}, q_0q_1^{'}, q_1q_0^{'}$ und $q_1q_1^{'}$.

Differenzautomaten können verwendet werden, um zu entscheiden, ob die Sprache eines Automaten in der Sprache des anderen Automaten komplett enthalten ist. Diese werden später dazu verwendet, um zu überprüfen, ob ein Prozessmodell im anderen enthalten ist, also ob z.B. ein Modell überspezifiziert ist. Dazu wird lediglich folgende Beobachtung von Wichtigkeit sein:

$$\mathcal{L}(M_i \setminus M_j) = \emptyset \iff \mathcal{L}(M_i) \subseteq \mathcal{L}(M_j)$$

Das bedeutet, dass die Sprache eines Automaten M_i in der Sprache eines anderen Automaten M_j genau dann enthalten ist, wenn der zugehörige Differenzautomat $M_i \setminus M_j$ die leere Sprache erkennt.

2.2.9 Symmetrische Differenz

Eines der Hauptziele der vorliegenden Arbeit ist das Vergleichen von Prozessmodellen. Dafür wird eine Methode zum Automatenvergleich benötigt, die die *symmetrische Differenz* zweier Automaten verwendet:

Definition 15. *Seien* $M_1 = (\Sigma, Q_1, q_{0_1}, \delta_1, F_1)$ *und* $M_2 = (\Sigma, Q_2, q_{0_2}, \delta_2, F_2)$ *DEAs über der gleichen Menge von Symbolen* Σ*. Dann heißt*

$$M_1 \Delta M_2 := (\Sigma, Q_1 \times Q_2, (q_{0_1}, q_{0_2}), \delta_{M_1 \Delta M_2}, F_{M_1 \Delta M_2})$$

mit $\delta_{M_1 \Delta M_2} : Q \times \Sigma \to Q, ((q_1, q_2), a) \mapsto (\delta_1(q_1, a), \delta_2(q_2, a))$ *und* $F_{M_1 \Delta M_2} = F_1 \times (Q_2 \setminus F_2) \cup (Q_1 \setminus F_1) \times F_2$ *die* ***symmetrische Differenz*** *von* M_1 *und* M_2*.*

$\delta_{M_1 \Delta M_2}$ unterscheidet sich vom Produktautomaten $M_1 \times M_2$ nur in der Menge der akzeptierenden Zustände und akzeptiert alle Wörter, die entweder nur von M_1 oder nur von M_2 akzeptiert werden.

Die Konstruktion der symmetrischen Differenz kann genutzt werden, um zu entscheiden, ob zwei DEAs M_1 und M_2 die gleiche Sprache erkennen. Dies ist genau dann der Fall, wenn das zugehörige symmetrische Produkt $M_1 \Delta M_2$ leer ist, d.h. kein Wort in der Sprache von $M_1 \Delta M_2$ liegt:

$$\mathcal{L}(M_1) = \mathcal{L}(M_2) \iff \mathcal{L}(M_1 \Delta M_2) = \emptyset$$

Nach Anwendung des Hopcroft Minimalisierungsalgorithmus ist obige Gleichung äquivalent zur Aussage, dass das symmetrische Produkt $M_1 \Delta M_2$ keinen akzeptierenden Zustand besitzt:

$$\mathcal{L}(M_1) = \mathcal{L}(M_2) \iff F_{M_1 \Delta M_2} = \emptyset$$

2.2.10 Übersicht

Alle in diesem Kapitel vorgestellten Konstrukte sind in Abbildung 2.14 als Venn-Diagramme dargestellt. Dabei kennzeichnet die schraffierte Menge jeweils die Sprache des betrachteten Automaten.

Der linke Kreis beschreibt die Sprache des Automaten M_1 (links oben in Abbildung 2.14), der rechte Kreis die Sprache von M_2 (rechts oben in Abbildung 2.14). Die mittlere Zeile von Abbildung 2.14 zeigt links den Schnitt dieser beiden Mengen, also die Sprache des Produktautomaten. Mit dessen Hilfe können Gemeinsamkeiten von M_1 und M_2 bestimmt werden. Rechts daneben ist die Sprache des symmetrischen Produkts von M_1 und M_2 dargestellt. Diese ist das Gegenteil zur Produktsprache und beschreibt die Wörter, die nur genau in einer der beiden Sprachen liegen. Basierend darauf können individuelle Verhaltensweisen der beiden Sprachen terminiert werden.

Die letzte Zeile von Abbildung 2.14 zeigt die Sprachen der beiden Differenzautomaten zu M_1 und M_2 mit deren Hilfe Wörter untersucht werden können, die ausschließlich von M_1 (oder M_2) akzeptiert werden. Diese Differenzautomaten werden in Kapitel 4 verwendet, um ungewolltes Verhalten von Prozessmodellen nach Modellanpassungen zu identifizieren.

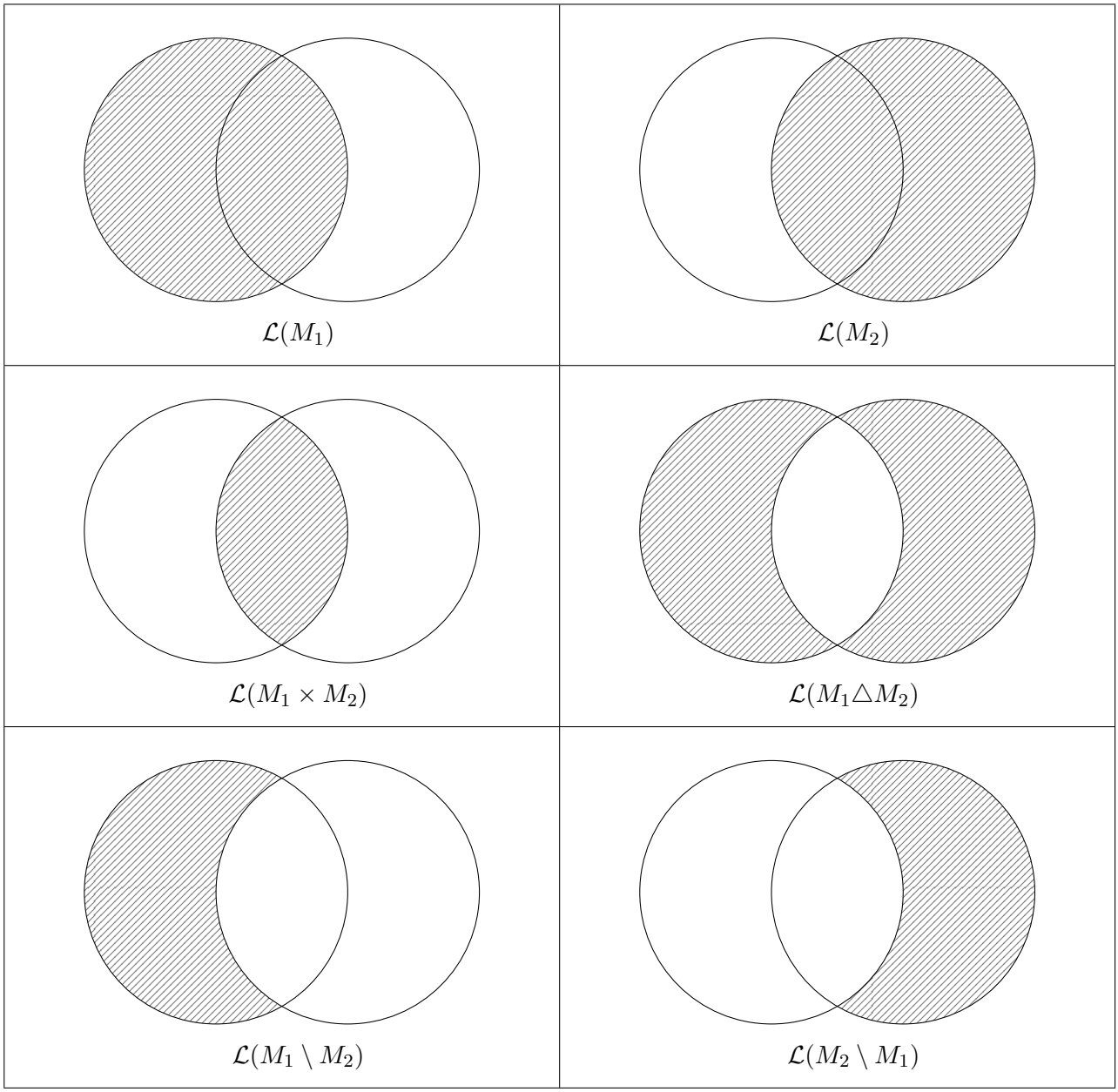


Abbildung 2.14: Venn-Diagramme zu DEA-Konstrukten

Kapitel 3

Vergleich von Prozessmodellen

In diesem Kapitel werden Prozessmodelle auf Gemeinsamkeiten und Unterschiede untersucht. Dies geschieht anhand der in Kapitel 2.1.3 beschriebenen Prozessperspektiven. Die *funktionale Perspektive* bildet dabei die Grundlage, um überhaupt erst den Vergleich bezüglich anderer Perspektiven zu ermöglichen. Die Standardisierung dieser Perspektive wird in Kapitel 3.1 erläutert. In vielen Bereichen der Forschung konzentrieren sich die theoretischen Untersuchungen von Prozessmodellen weitgehend auf die *funktionale Perspektive* und insbesondere die *Kontrollflussperspektive*. Daran hält sich auch die vorliegende Dissertation und deshalb wird der Modellvergleich im Hinblick auf diese Perspektive auch den größten Teil der Arbeit ausmachen. Dieser wird in Kapitel 3.2 vorgestellt. Anschließend werden noch Vergleichsmethoden hinsichtlich der *organisatorischen Perspektive* (Kapitel 3.3.1), der *operationalen Perspektive* (Kapitel 3.3.2) sowie der *Daten- und Datenflussperspektive* (Kapitel 3.3.3) entwickelt. Abschnitt 3.3.4 interpretiert die entwickelten Ansätze und gibt Empfehlungen für deren Anwendung. Abschließend wird in Kapitel 3.4 ein Überblick über in der Literatur bereits existierende Ansätze zum Vergleich von Prozessmodellen gegeben.

3.1 Standardisierung und Abgleich der funktionalen Perspektive

Zunächst werden die Standardisierung und der Abgleich der *funktionalen Perspektive* (Kapitel 2.1.3) betrachtet (erster Schritt (*Standardisierung der funktionalen Perspektive*) des Gesamtkonzepts in Abbildung 1.5). Mit ihrer Hilfe werden Prozessmodelle im Grunde erst vergleichbar gemacht. Standardisierung der *funktionalen Perspektive* bedeutet, dass gleiche Elemente in zwei Prozessmodellen gleich benannt sein müssen. Dieser Schritt ist notwendig für sämtliche zukünftigen Vergleiche in Bezug auf weitere Perspektiven. Beispielsweise ist die einheitliche Benennung der Aktivitäten der beiden Prozesse Voraussetzung für den anschließenden Vergleich der *Kontrollflussperspektive*, da diese Aktivitäten die Zustandsübergänge von endlichen deterministi-

schen Automaten bilden werden (Kapitel 3.2) und deshalb die Sequenzen der Prozessmodelle als Wörter über den Symbolen der Aktivitäten interpretiert werden. Eine uneinheitliche Benennung der Aktivitäten bzw. der Zustandsübergangssymbole würde somit keinen Vergleich der zu Grunde liegenden Sprachen ermöglichen. Ebenso müssen alle Elemente der *organisatorischen Perspektive*, z.B. Personen, in beiden Prozessmodellen gleich benannt sein. Analoges gilt für alle beteiligten Elemente der *operationalen Perspektive* sowie der *Daten- und Datenflussperspektive*.

Der direkte Abgleich der funktionalen Perspektive ist nicht Teil der vorliegenden Dissertation, da dieser außerhalb des eigentlichen Prozessmodellvergleichs durchzuführen ist. Es wird im Verlauf dieser Arbeit davon ausgegangen, dass das beschriebene Matching aller Elemente bereits durchgeführt wurde. Dies wäre zum Beispiel möglich durch das Hinzunehmen einer Domänenexpertin oder eines Domänenexperten oder das Anwenden von Verfahren aus dem Bereich des *Natural Language Processing (NLP)*, z.B. [52, 53, 54]. In den weiteren Überlegungen wird vorausgesetzt, dass die beiden zu vergleichenden Prozessmodelle bezüglich der *funktionalen Perspektive* gleich sind.

3.2 Vergleich der Kontrollflussperspektive

Die nun folgenden Ausführungen befassen sich mit dem Vergleich von Prozessmodellen hinsichtlich der *Kontrollflussperspektive* (Kapitel 2.1.3). Um Prozessmodelle, die in verschiedenen Sprachen modelliert sind, bezüglich dieser Perspektive untersuchen zu können, wird eine gemeinsame Grundlage benötigt. Diese Grundlage bilden in dieser Arbeit sogenannte *Prozessautomaten*. Dabei handelt es sich um endliche deterministische Automaten (Kapitel 2.2.1), die das Verhalten der Prozessmodelle beschreiben. Diese Prozessautomaten werden in Kapitel 3.2.1 eingeführt. Anschließend werden Transformationen von Declare-Modellen (Kapitel 3.2.2) und BPMN-Modellen (Kapitel 3.2.3) in Prozessautomaten angegeben. Zum eigentlichen Prozessmodellvergleich hinsichtlich der Kontrollflussperspektive werden danach zwei verschiedene Methoden vorgestellt: der *simulationsbasierte Ansatz* (Kapitel 3.2.4 und 3.2.5) sowie der *inklusionsbasierte Ansatz* (Kapitel 3.2.6). Außerdem werden Methoden zur Überprüfung von Prozessmodellen auf Teilmengenbeziehungen (Kapitel 3.2.7) sowie zur Berechnung von Gemeinsamkeiten (Kapitel 3.2.8) und Unterschieden (Kapitel 3.2.9) von Prozessmodellen bezüglich der Kontrollflussperspektive vorgestellt. Abbildung 3.1 zeigt, dargestellt als BPMN-Modell, das komplette Vorgehen zum Vergleich des Kontrollflusses zweier Prozessmodelle. Alle in diesem Kapitel beschriebenen Ansätze und Methoden basieren auf den Publikationen [11, 21, 22] des Autors.

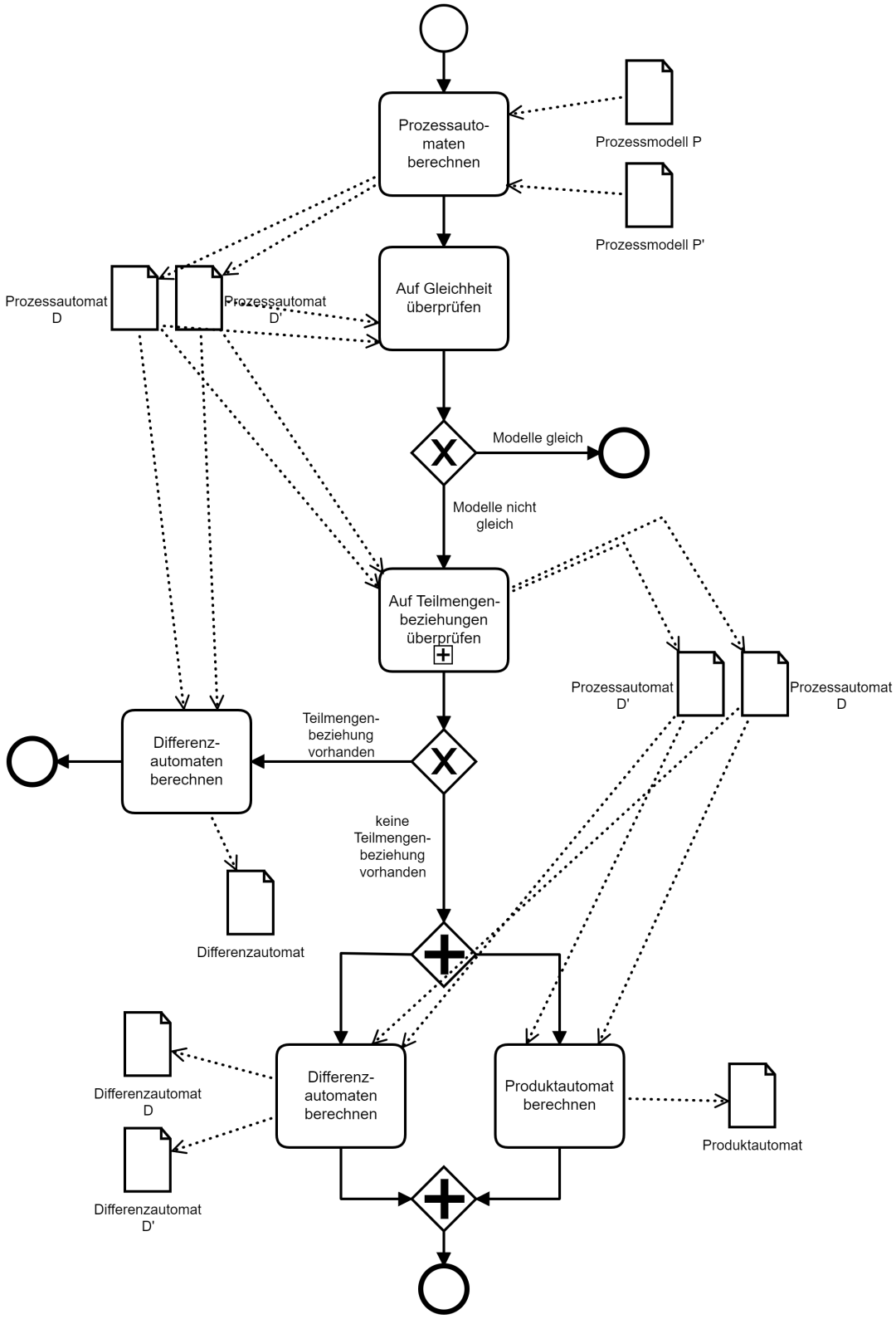


Abbildung 3.1: Gesamtkonzept zum Vergleich der Kontrollflussperspektive

$\delta_1(q_0, A) = q_1$	$\delta_1(q_1, A) = q_1$	$\delta_1(q_2, A) = q_2$
$\delta_1(q_0, B) = q_2$	$\delta_1(q_1, B) = q_1$	$\delta_1(q_2, B) = q_2$

Tabelle 3.1: Zustandsüberführungsfunktion δ_1

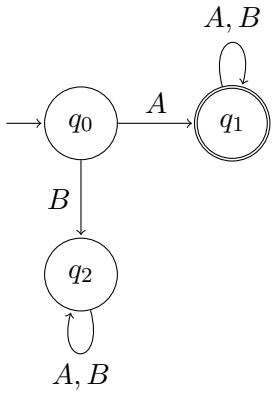


Abbildung 3.2: Graphische Repräsentation von M_1

3.2.1 Prozessautomaten und Gleichheit von Prozessmodellen

In diesem Kapitel werden sogenannte *Prozessautomaten* definiert, die die Grundlage für den ersten Schritt (*Prozessautomaten berechnen*) in Abbildung 3.1 bilden. Diese sind eine äquivalente Repräsentation eines Prozessmodells. Ein Prozessautomat ist ein deterministischer endlicher Automat (Kapitel 2.2.1), bei dem die Zustandsübergänge über die Aktivitäten des Prozesses definiert werden:

Definition 16. *Ein **Prozessautomat** ist ein DEA $P = (\mathcal{A}, Q, q_0, \delta, F)$, wobei $\mathcal{A}$ eine endliche (nicht-leere) Menge an Symbolen ($\widehat{=}$ Aktivitäten), Q eine endliche (nicht-leere) Menge von Zuständen, $q_0 \in Q$ ein Startzustand, $\delta : Q \times \Sigma \to Q$ eine erweiterte Zustandsüberführungsfunktion und $F \subseteq Q$ eine Menge von Endzuständen ist.*

Der Startzustand q_0 steht für einen Prozess vor dessen Ausführung. Das Ausführen einer Aktivität $A \in \mathcal{A}$ entspricht einem Zustandsübergang im zugehörigen Automaten, der mit A beschriftet ist. Endzustände geben an, dass die bisher verarbeitete Sequenz (Kapitel 2.1.1) eine gültige Sequenz des Prozessmodells ist. Analog bedeutet ein Zustand, der kein Endzustand ist, dass die bisher verarbeitete Sequenz keine gültige Sequenz des Prozessmodells ist.

Gegeben sei DEA M_1 aus Kapitel 2.2: $M_1 = (\{A, B\}, \{q_0, q_1, q_2\}, q_0, \delta_1, \{q_1\})$, wobei δ_1 in Tabelle 3.1 angegeben ist. Die graphische Repräsentation von M_1 ist nochmals in Abbildung 3.2 dargestellt. Interpretiert man M_1 als Prozessautomaten, bedeutet dies, dass der zu Grunde liegende Prozess aus zwei Aktivitäten ($\mathcal{A} = \{A, B\}$) besteht. M_1 akzeptiert alle Wörter, die mit

A beginnen (Kapitel 2.2). Somit sind für das zugehörige Prozessmodell alle Sequenzen, die mit A beginnen, gültige Sequenzen.

Zusammengefasst bedeutet die Definition eines Prozessautomaten: Aktivitäten eines Prozesses werden mit den Symbolen des Prozessautomaten, Sequenzen mit Wörtern und das Ausführen von Aktivitäten mit der Zustandsüberführung identifiziert.

Mit Hilfe der Prozessautomaten kann nun die Gleichheit für Prozessmodelle definiert werden. Dabei wird in dieser Arbeit der Gleichheitsbegriff aus [12] verwendet, der in Kapitel 1.2 bereits informell eingeführt wurde. Dieser identifiziert zwei Prozessmodelle als „gleich“, wenn sie die gleichen Sequenzen akzeptieren:

Definition 17. *Seien P_1 und P_2 Prozessmodelle mit zugehörigen Prozessautomaten M_1 und M_2. Dann heißen P_1 und P_2* ***gleich (bezüglich der Kontrollflussperspektive)****, wenn $\mathcal{L}(M_1) = \mathcal{L}(M_2)$.*

Im weiteren Verlauf dieser Dissertation wird für ein Prozessmodell P mit zugehörigem Prozessautomaten M die Sprache von P mit $\mathcal{L}(P) := \mathcal{L}(M)$ bezeichnet.

3.2.2 Transformation eines Declare-Modells in einen Prozessautomaten

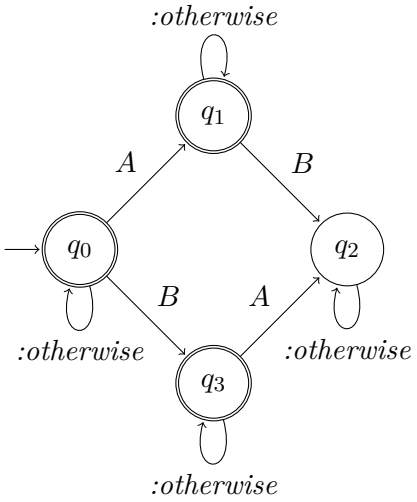
Im Folgenden wird eine Transformation eines Declare-Modells (Kapitel 2.1.4) in einen Prozessautomaten (Kapitel 3.2.1) beschrieben [21, 22]. Diese bildet die Grundlage des in dieser Arbeit entwickelten Ansatzes zum Vergleich von Prozessmodellen, da dieser auf Automatentheorie (Kapitel 2.2) basiert (erster Schritt (*Prozessautomaten berechnen*) in Abbildung 3.1). Declare wird in dieser Arbeit als Repräsentant einer deklarativen Modellierungssprache verwendet, da diese Sprache sehr gut erforscht. Die im Folgenden verwendeten Automatendarstellungen basieren auf [21, 22, 55].

Der erste Schritt bei der Modelltransformation eines Declare-Prozessmodells $P = (\mathcal{A}, \mathcal{T})$ in einen endlichen deterministischen Automaten ist die Transformation der Constraints $\tau \in \mathcal{T}$ in einen sogenannten *Constraint-Automaten* M_τ:

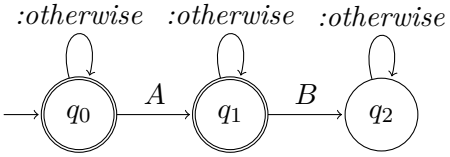
Definition 18. *Sei $\mathcal{A}$ eine endliche Menge von Aktivitäten und τ ein Declare Constraint über $\mathcal{A}$. Der zugehörige* ***Constraint-Automat*** *M_τ zu τ ist definiert als der minimale endlich deterministische Automat, sodass*

$$\mathcal{L}(M_\tau) = \{t \text{ Sequenz über } \mathcal{A} \mid t \text{ ist nicht von } \tau \text{ verboten}\}.$$

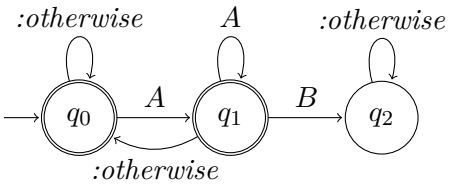
Mit anderen Worten: M_τ ist der korrespondierende Prozessautomat zu einem Declare-Modell $P_\tau = (\mathcal{A}, \{\tau\})$, das nur aus einem einzigen Constraint, nämlich τ, besteht. Die Sprache $\mathcal{L}(M_\tau)$ des Prozessautomaten M_τ besteht folglich aus allen Sequenzen, die das Prozessmodell erfüllen.



(a) notRespondedExistence(A, B)



(b) notResponse(A, B)



(c) notChainResponse(A, B)

Abbildung 3.3: Constraint-Automaten

Die Transformation der meisten Declare-Constraints in Constraint-Automaten wird bereits in [55] erledigt. Lediglich die drei Constraints notRespondedExistence(A, B), notResponse(A, B) und notChainResponse(A, B) fehlen. Diese werden analog berechnet und sind in Abbildung 3.3 dargestellt [22]. Die Constraint-Automaten zu allen in dieser Arbeit verwendeten Declare-Constraints (Kapitel 2.1.4) sind in Anhang A.1 zu finden.

Man sieht, dass einige Zustandsübergänge mit *:otherwise* beschriftet sind. Dies liegt daran, dass diese Übergänge von der zu Grunde liegenden Menge an Aktivitäten $\mathcal{A}$ abhängig sind, denn Constraints beinhalten im Regelfall nur maximal zwei Aktivitäten. Natürlich muss aber auch die Ausführung aller anderen Aktivitäten berücksichtigt werden. Beispielsweise würde die Ausführung einer Aktivität C auf ein Constraint wie response(A, B) keinen Einfluss haben. Allerdings hätte sie das unter Umständen beim chainResponse(A, B) Constraint, da die Ausführung von C direkt nach der Ausführung von Aktivität A nicht erlaubt ist. Somit ist für alle Aktivitäten eine Zustandsüberführung zu definieren. Die Zustandsüberführungsbeschriftung *:otherwise* aus einem Zustand q steht demnach stellvertretend für alle Aktivitäten, die noch keine Zustandsüberführung aus q besitzen. Dies bedeutet beispielhaft für das notRespondedExistence(A, B) Constraint

(Abbildung 3.3a): Wäre die zu Grunde liegende Menge an Aktivitäten $\{A, B, C\}$, würde die reflexive Zustandsüberführung statt mit *:otherwise* mit C beschriftet werden. Für den Fall, dass die Menge der Aktivitäten aus $\{A, B, C, D\}$ besteht, würde *:otherwise* durch C, D ersetzt werden.

Nachdem Constraint-Automaten für einzelne Declare Constraints definiert wurden, kann nun zur Transformation eines Declare-Modells $P = (\mathcal{A}, \mathcal{T} = \{\tau_1, \ldots, \tau_n\})$ in einen DEA übergegangen werden. Zunächst sei angemerkt, dass eine Sequenz σ das Declare-Modell P genau dann erfüllt, falls sie alle Constraints $\tau_i \in \mathcal{T}$ erfüllt:

$$\sigma \text{ erfüllt } P \Longleftrightarrow \sigma \text{ erfüllt } \tau_1 \wedge \cdots \wedge \sigma \text{ erfüllt } \tau_n$$

Auf Grund der obigen Überlegungen ist die rechte Bedingung äquivalent zur Aussage, dass σ in der Sprache aller Constraint-Automaten M_{τ_i} liegt, d.h. $\sigma \in \mathcal{L}(M_{\tau_i})$ für alle $i \in \{1, ..., n\}$. Damit erhält man:

$$\sigma \text{ erfüllt } P \Longleftrightarrow \sigma \in \mathcal{L}(M_{\tau_1}) \wedge \cdots \wedge \sigma \in \mathcal{L}(M_{\tau_n}) \Longleftrightarrow \sigma \in \bigcap_{i \in \{1,\ldots,n\}} \mathcal{L}(M_{\tau_i})$$

Somit bestehen alle von P akzeptierten Sequenzen aus den Sequenzen, die im Durchschnitt der Sprachen der Constraint-Automaten liegen. Dieser Durchschnitt kann mit Hilfe des Produktautomaten (Kapitel 2.2.6) berechnet werden. Daraus ergibt sich, dass P genau die Sequenzen akzeptiert, die vom Produktautomaten $M_{\mathcal{T}} := \prod_{\tau_i \in \mathcal{T}} M_{\tau_i}$ akzeptiert werden:

$$\sigma \text{ erfüllt } P \Longleftrightarrow \sigma \in \mathcal{L}(M_{\mathcal{T}})$$

Der Automat $M_{\mathcal{T}}$ ist dann der zu P korrespondierende Prozessautomat. Dieser besitzt $\prod_{\tau_i \in \mathcal{T}} |M_{\tau_i}|$ Zustände, wobei mit $|M_{\tau_i}|$ die Anzahl der Zustände des Constraint-Automaten M_{τ_i} bezeichnet wird.

Um die Performance der Anwendung und den Speicherbedarf zu optimieren, ist es wünschenswert, Automaten so klein wie möglich zu halten. Deshalb wird in dieser Arbeit jeweils nach dem Produkt von zwei Automaten der resultierende Automat minimiert (Kapitel 2.2.7). Diese Minimierung ändert nichts am Akzeptanzverhalten des Automaten, sondern reduziert lediglich die Anzahl der Zustände. Das komplette Vorgehen zur Transformation eines Declare-Modells in einen Prozessautomaten ist in Algorithmus 1 dargestellt.

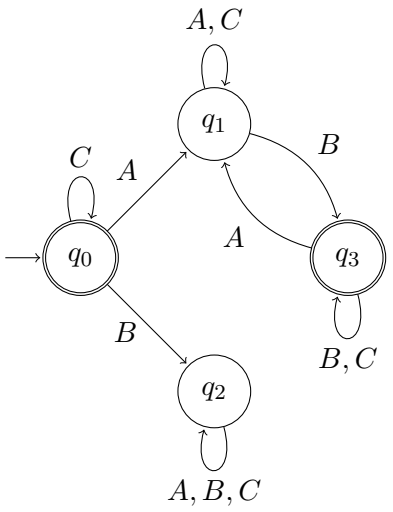
Im Folgenden wird zur Illustration der Transformation der Prozessautomat zu folgendem Declare-Prozessmodell P berechnet: $P = (\{A, B, C\}, \mathcal{T})$ [22], wobei $\mathcal{T} = \{\text{succession}(A, B)$, $\text{chainResponse}(A, B)$, $\text{chainResponse}(B, C)$, $\text{respondedExistence}(A, B)\}$. Zuerst wird zu jedem Constraint der zugehörige Constraint-Automat berechnet. Abbildung 3.4 zeigt die beiden Constraint-Automaten $M_{\text{succession}(A,B)}$ und $M_{\text{chainResponse}(A,B)}$. Die beiden anderen Constraint-

Algorithmus 1: *calculateDeclareProcessAutomaton*

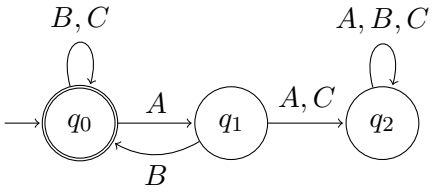
Input: Declare Process Model $P = (\mathcal{A}, \mathcal{T})$
Output: Process Automaton M_P representing P

$U \leftarrow \emptyset, i \leftarrow 1$
for $\tau \in \mathcal{T}$ **do**
  $(\mathcal{A}, S_\tau, s_{0_\tau}, \delta_\tau, F_\tau) \leftarrow$ transform τ into constraint automaton
  $\tau_i \leftarrow (\mathcal{A}, S_\tau, s_{0_\tau}, \delta_\tau, F_\tau)$
  $U \leftarrow U.\text{add}(\tau_i)$
  $i \leftarrow i + 1$
end
for $j = 1, \ldots, i - 1$ **do**
  $\tau_{j+1} \leftarrow \text{minimization}(\text{product}(\tau_j, \tau_{j+1}))$
end
$M_P \leftarrow \tau_{j+1}$
return M_P

Automaten sind in Anhang A.2 zu finden.



(a) $M_{\text{succession}(A,B)}$



(b) $M_{\text{chainResponse}(A,B)}$

Abbildung 3.4: Constraint-Automaten $M_{\text{succession}(A,B)}$ und $M_{\text{chainResponse}(A,B)}$

Nun wird der Produktautomat $M_{\text{succession}(A,B)} \times M_{\text{chainResponse}(A,B)}$ von $M_{\text{succession}(A,B)}$ und $M_{\text{chainResponse}(A,B)}$ berechnet. Dieser ist in Abbildung 3.5 dargestellt. Dieser Produktautomat ist offensichtlich noch nicht minimal, da er beispielsweise mit q_1 und q_3 unerreichbare Zustände besitzt. Nach Minimierung dieses Automaten (und Umbenennen der Zustände) ergibt sich der in Abbildung 3.6 dargestellte DEA.

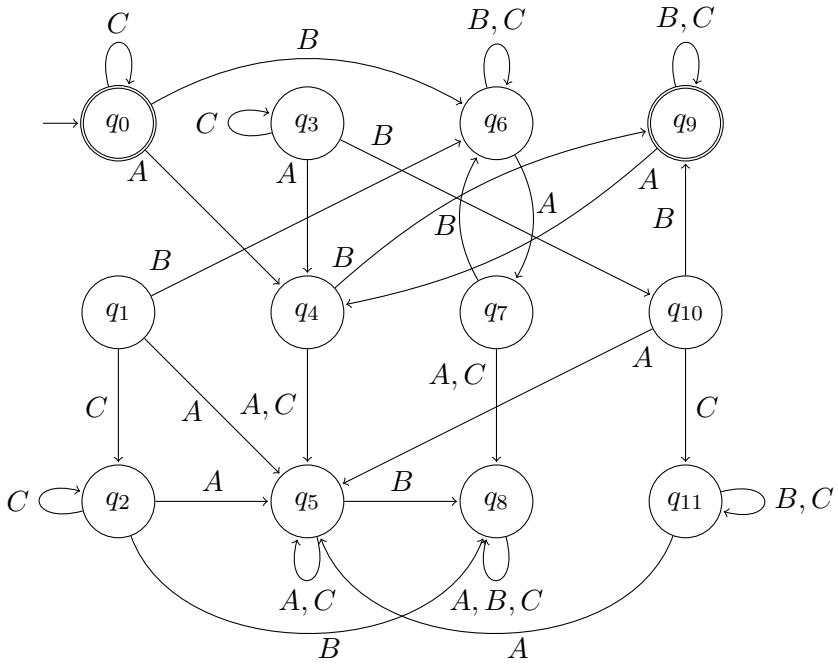


Abbildung 3.5: Produktautomat $M_{\text{succession}(A,B)} \times M_{\text{chainResponse}(A,B)}$

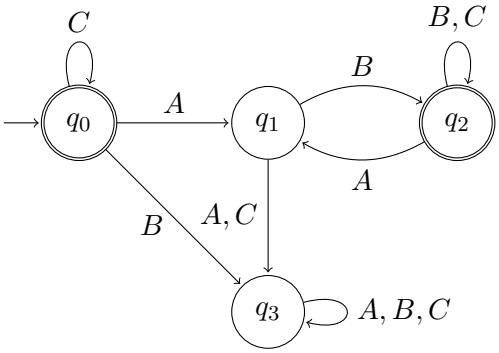


Abbildung 3.6: Minimierter Produktautomat von $M_{\text{succession}(A,B)} \times M_{\text{chainResponse}(A,B)}$

Das Hinzumultiplizieren der beiden weiteren Constraint-Automaten $M_{\text{chainResponse}(B,C)}$ und $M_{\text{respondedExistence}(A,B)}$ und das Minimieren der korrespondierenden Automaten erfolgen analog und sind ebenfalls in Anhang A.2 zu finden. Schließlich erhält man den in Abbildung 3.7 skizzierten Prozessautomaten für P.

3.2.3 Transformation eines BPMN-Modells in einen Prozessautomaten

Nach der Vorstellung der Transformation eines Declare-Modells in einen Prozessautomaten (Kapitel 3.2.2) als Repräsentant einer deklarativen Prozessmodellierungssprache wird nun eine

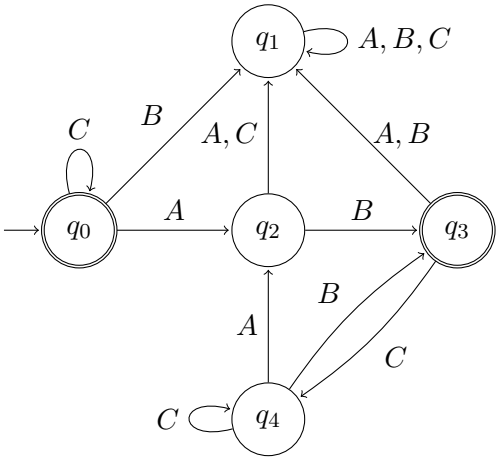


Abbildung 3.7: Prozessautomat zu P

ähnliche Transformation für imperative Prozessmodelle, speziell für BPMN-Modelle (Kapitel 2.1.4), vorgestellt (erster Schritt (*Prozessautomaten berechnen*) in Abbildung 3.1) [11]. Dieser Schritt wird es ermöglichen, imperative und deklarative Modelle auf einer gemeinsamen Basis zu vergleichen und Aussagen über Gemeinsamkeiten und Unterschiede der Prozessmodelle zu treffen.

Für die Transformation eines BPMN-Modells in einen DEA wird das BPMN-Modell in Teilgraphen unterteilt. Dabei werden zwei verschiedene Arten von Teilgraphen betrachtet. Diese werden im Folgenden formal definiert [11].

Definition 19. *Ein zusammenhängender Teilgraph* $G = (F', E', E^{s'}, E^{e'}, A', G', G^{p'}, G^{e'}, T') \subseteq P$ *eines BPMN-Modells* $P = (F, E, E^s, E^e, A, G, G^p, G^e, T)$ *heißt* maximale Aktivitätssequenz*, falls*

1. $F' = \{A_1, \ldots, A_n \mid A_i \in A\} \subseteq A$,
2. $\exists! A_i \in F' : (X, A_i) \in T \wedge X \in F \setminus A$,
3. $\exists! A_j \in F' : (A_j, X) \in T \wedge X \in F \setminus A$.

Das bedeutet, dass eine Aktivitätssequenz ausschließlich aus Aktivitäten besteht (Punkt 1 in der Definition). Zusätzlich wird gefordert, dass genau eine Aktivität eine eingehende Transition mit einem Flow-Objekt, das keine Aktivität ist, besitzt (Punkt 2 in der Definition). Der dritte Punkt der Definition fordert, dass genau eine Aktivität eine ausgehende Transition besitzt, die nicht in einer anderen Aktivität endet.

Abbildung 3.8 zeigt ein BPMN-Prozessmodell, das eine Aktivitätssequenz der Länge drei enthält. Die zugehörige Aktivitätssequenz besteht aus drei Aktivitäten (A, B, C) sowie den dazwischenliegenden Transitionen. Aktivität A erfüllt dabei die zweite Bedingung der Definition,

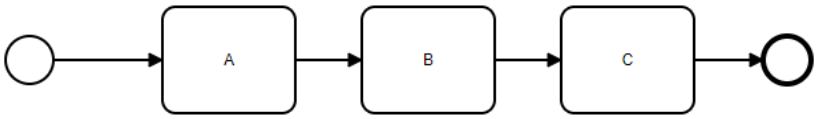


Abbildung 3.8: BPMN-Prozessmodell mit maximaler Aktivitätssequenz der Länge 3

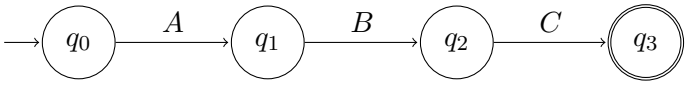


Abbildung 3.9: DEA für Aktivitätssequenz aus Abbildung 3.8

da die eingehende Transition aus dem Start-Event des Modells kommt. Aktivität C besitzt eine ausgehende Transition, die im End-Event des Modells endet, und erfüllt somit die dritte Bedingung der Definition.

Um eine Aktivitätssequenz in einen DEA zu transformieren, wird sukzessive ein Automat gebaut, der genau diese Sequenz repräsentiert. Für eine Aktivitätssequenz der Länge n entsteht dann ein DEA, der aus $n - 1$ Zuständen besteht. Diese sind sequentiell durch Transitionen verbunden, die die Aktivitätsnamen der Aktivitätssequenz als Übergänge besitzen. Zudem gibt es noch jeweils eine weitere Transition aus jedem Zustand, die in den Totzustand q_d führt, falls nicht die gewünschte Aktivität ausgeführt wird. Abbildung 3.9 zeigt beispielhaft den DEA für das BPMN-Modell aus Abbildung 3.8 (der Totzustand q_d wurde aus Übersichtsgründen weggelassen).

In einem nächsten Schritt werden Gateway-Hierarchien eingeführt, die bei der Transformation von verschachtelten BPMN-Modellen in DEAs benötigt werden. Es lässt sich feststellen, dass durch BPMN-Modelle implizit Gateway-Hierarchien gegeben sind. Abbildung 3.10 zeigt drei Gateways (jeweils drei öffnende und drei schließende): Dabei umfasst ein exklusives Gateway ($g1$) ein paralleles Gateway ($g2$). Daraus lässt sich der Hierarchie-Baum in Abbildung 3.11 ableiten. Die Blattknoten (hier die Knoten für $g2$ und $g3$) stehen dabei für die „äußersten“ Gateways. Diese werden benötigt, um die Transformation in DEAs zu realisieren.

Zunächst wird davon ausgegangen, dass das zu transformierende BPMN-Modell keine Schleife besitzt. Gateways werden sukzessive transformiert. Das bedeutet, dass grundsätzlich mit den „inneren“ Gateway-Paaren (d.h. den Blättern in der zugehörigen Hierarchie) begonnen und dann weiter „nach außen“ gegangen wird (d.h. eine Hierarchieebene höher). Damit gilt, dass innerhalb eines „inneren Gateways“ immer maximale Aktivitätssequenzen (siehe oben) auftreten. Somit können mit Hilfe der Transformationsregel für Aktivitätssequenzen die Regeln zur Transformation von Gateways angegeben werden:

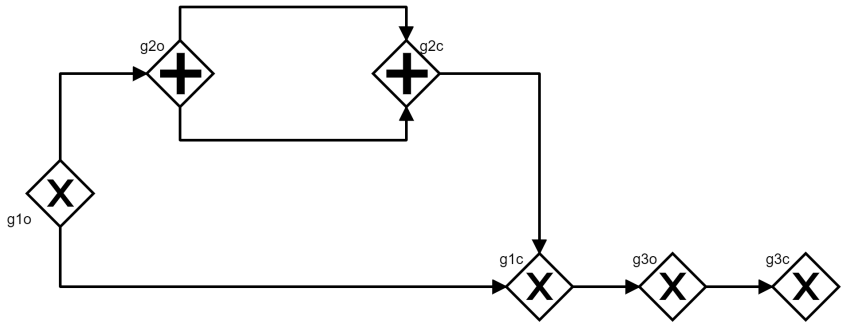


Abbildung 3.10: BPMN-Modell bestehend aus drei Gateway-Paaren

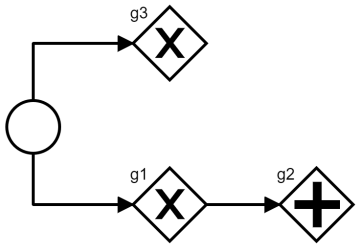


Abbildung 3.11: Gateway-Hierarchie zu Abbildung 3.10

- Ein exklusives Gateway mit n ausgehenden Kanten wird durch einen neuen Zustand q_e repräsentiert. Des Weiteren wird für alle ausgehenden Kontrollflüsse die Regel für maximale Aktivitätssequenzen angewandt. Diese neuen DEAs $A_1, \ldots, A_n$ werden dann jeweils an den Zustand q_e konkateniert.

- Ein paralleles Gateway mit n ausgehenden Kanten wird durch einen neuen Zustand q_p repräsentiert. Danach werden zunächst die Automaten $A_1, \ldots, A_n$ für die Aktivitätssequenzen berechnet (wie bei exklusiven Gateways). Im Gegensatz zu den exklusiven Gateways werden hier die Automaten aber nicht einfach konkateniert. Das Alphabet Σ_i eines jeden Automaten A_i wird allerdings auf die Vereinigung $\Sigma := \bigcup_{i=\{1,\ldots,n\}} \Sigma_i$ erweitert, da in jedem Automat alle vorkommenden Aktivitäten erlaubt sein müssen. Zudem wird in jedem Automaten A_i in jedem Zustand eine reflexive Kante für alle Aktivitäten $\omega \in \Sigma \setminus \Sigma_i$ eingeführt. Die so entstandenen Automaten $A_1^*, \ldots, A_n^*$ werden nun sukzessive multipliziert. Der Automat $A := \prod_{\{i=1,\ldots,n\}} A_i^*$ repräsentiert schließlich das parallele Gateway.

Abbildung 3.2 zeigt in einer Übersicht die Transformationsregeln für Aktivitätssequenzen, exklusive Gateways und parallele Gateways. Die gestrichelten Elemente deuten jeweils den Konkatenationspunkt für die darauf folgenden Automaten an.

Abschließend wird noch der Fall betrachtet, dass ein BPMN-Diagramm eine Schleife besitzt.

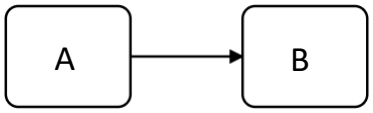
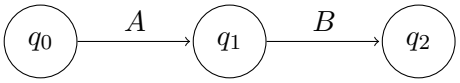
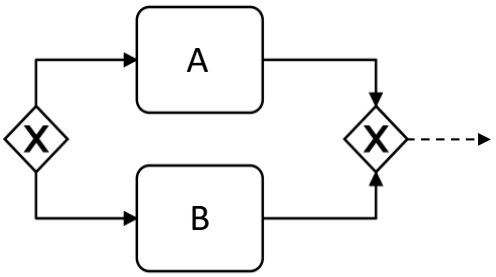
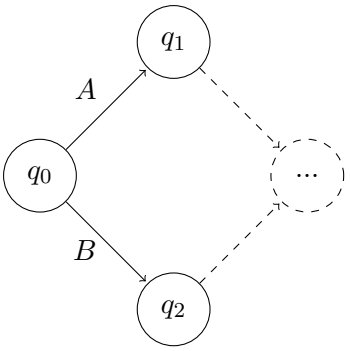
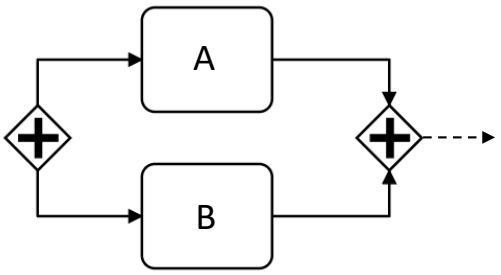
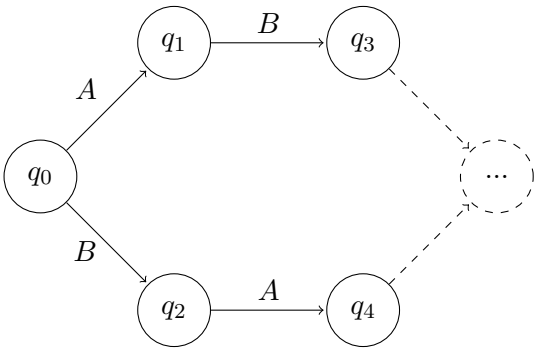
BPMN-Konstrukt	Transformationsregel
	
	
	

Tabelle 3.2: Transformationsregeln für BPMN-Konstrukte [11]

Sei dafür ein BPMN-Modell P mit umfassendem Gateway-Paar (g_0, g_1) gegeben. Eine Schleife bedeutet, dass es ausgehend von g_1 einen Teilgraphen G_0 gibt, der in g_0 führt. Weiter seien alle von g_0 ausgehenden Teilgraphen mit $G_1, \ldots, G_n$ bezeichnet. Für die Transformation wird nun folgendermaßen vorgegangen:

1. Konstruiere die DEAs $A_0, A_1, \ldots, A_n$ zu den Teilgraphen $G_0, G_1, \ldots, G_n$.
2. Kombiniere $A_1, \ldots, A_n$ zu einem DEA A wie bei der oben beschriebenen Transformation der Gateways.
3. Konstruiere den DEA A mit $\mathcal{L}(A) = \mathcal{L}(A \circ A_0)^*$.

Abbildung 3.12 zeigt ein BPMN-Modell (links) und den zugehörigen Prozessautomaten (rechts), der durch Anwenden der Transformationsregeln entsteht.

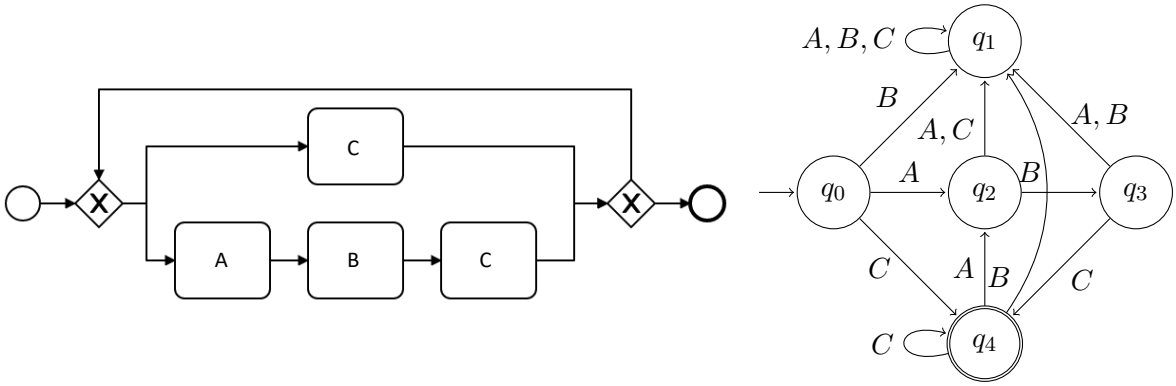


Abbildung 3.12: BPMN-Modell (links) [11] und zugehöriger Prozessautomat (rechts) [11]

3.2.4 Simulationsbasierter Ansatz

In diesem Abschnitt wird eine Methode zum Vergleich von endlich deterministischen Automaten eingeführt, die zum Vergleich von Prozessmodellen bezüglich der Kontrollflussperspektive verwendet werden kann (zweiter Schritt (*Auf Gleichheit überprüfen*) in Abbildung 3.1). Wichtigster Bestandteil dieses Ansatzes ist die Berechnung der sogenannten *upper bound* [21]:

Satz 3. *Seien* $M_1 = (\Sigma, Q_1, q_{0_1}, \delta_1, F_1)$ *und* $M_2 = (\Sigma, Q_2, q_{0_2}, \delta_2, F_2)$ *DEAs über der gleichen Menge von Symbolen* Σ *und* $b := |Q_1| \cdot |Q_2|$*. Dann gilt:*

$$\mathcal{L}(M_1) = \mathcal{L}(M_2) \iff \{\omega \in \mathcal{L}(M_1) \mid |\omega| < b\} = \{\omega \in \mathcal{L}(M_2) \mid |\omega| < b\}$$

Bevor dieser Satz bewiesen wird, soll eine kurze Interpretation gegeben werden. Satz 3 besagt, dass die Sprachen von zwei DEAs genau dann gleich sind, wenn sie auf der Menge aller Wörter der Länge kleiner als das Produkt der Anzahl der Zustände der beiden Automaten übereinstimmen. Dieses Produkt $b := |Q_1| \cdot |Q_2|$ wird als *upper bound* bezeichnet. Nun zum Beweis des Satzes:

Beweis. *Die Hinrichtung* $\Rightarrow$ *ist trivial: falls die beiden Sprachen* $\mathcal{L}(M_1)$ *und* $\mathcal{L}(M_2)$ *gleich sind, stimmen* M_1 *und* M_2 *natürlich auch auf allen Teilmengen überein. Somit stimmen sie insbesondere auf der Menge der Wörter der Länge kleiner b überein.*

Die Rückrichtung $\Leftarrow$ *wird durch Kontraposition bewiesen. Angenommen es gilt* $\mathcal{L}(M_1) \neq \mathcal{L}(M_2)$*. Dann gibt es ein Wort* $a \in \Sigma^*$ *mit minimaler Länge, sodass* $a \notin \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$*. Sei weiter* $M := M_1 \times M_2$ *der Produktautomat (Kapitel 2.2.6) von* M_1 *und* M_2*. Da* $a \notin \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$*, folgt, dass* $a \notin \mathcal{L}(M)$*.*

Es wird durch Kontraposition angenommen, dass $|a| \geq b$ *gilt. Sei* $X := \{\delta_M(q_0, c) \mid$ *c ist Präfix von a*$\}$*. Da der Produktautomat M aus b Zuständen besteht, d.h.* $|Q_M| = b$*, und* $|X| \geq b + 1$*, gibt es zwei (verschiedene) Präfixe u und* u' *von a, sodass* $\delta_M(q_0, u) = \delta_M(q_0, u')$*.*

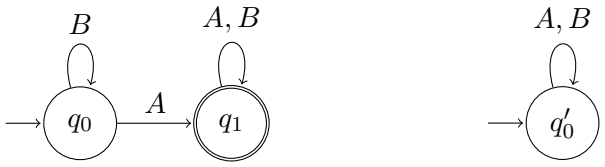


Abbildung 3.13: Zwei DEAs M_1 (links) und M_2 (rechts) mit $\mathcal{L}(M_1) \neq \mathcal{L}(M_2)$

Sei ohne Einschränkung angenommen, dass u ein Präfix von u′ ist. Das bedeutet, dass es ein Wort v gibt, sodass uv = u′. Da u′ ein Präfix von a ist, gibt es weiter ein Wort z, sodass u′z = a. Es folgt:

$$uvz = u'z = a$$

Aus u ≠ u′ folgt, dass v nicht leer ist. Also:

$$\delta_M(q_0, u) = \delta_M(q_0, u') = \delta_M(q_0, uv) = \delta_M(\delta_M(q_0, u), v)$$

Somit führt das Lesen von v im Zustand $\delta_M(q_0, u)$ durch eine Schleife wieder in den gleichen Zustand $\delta_M(q_0, u)$. Das bedeutet, dass $\delta_M(q_0, uz) = \delta_M(q_0, uvz) = \delta_M(q_0, a)$. Da aber v nicht leer ist, folgt, dass $|uz| < |a|$. Dies ist ein Widerspruch zur Minimalität von a. □

Die *upper bound b* ist eine „scharfe“ Grenze. Das bedeutet, dass es Fälle gibt, in denen zwei Automaten auf allen Wörtern bis zur Wortlänge $b-2$ übereinstimmen, sich allerdings bei Wörtern der Länge $b-1$ unterscheiden. Dies ist an den beiden Automaten M_1 und M_2 über dem Alphabet $\{A, B\}$ aus Abbildung 3.13 zu erkennen. Da M_1 beispielsweise das Wort A akzeptiert, M_2 aber gar kein Wort akzeptiert, gilt $\mathcal{L}(M_1) \neq \mathcal{L}(M_2)$. Die *upper bound* ist $b = 2 \cdot 1 = 2$. Beide Automaten stimmen auf allen Wörtern der Länge $b-2 = 0$ überein, da beide das leere Wort nicht akzeptieren. Unterschiede in beiden Automaten treten hier erst bei der Wortlänge $b-1 = 1$ auf, da sich M_1 und M_2, wie oben beschrieben, in der Akzeptanz des Wortes A unterscheiden.

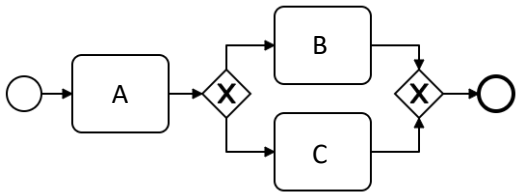
Satz 3 kann in folgender Weise zum Vergleich von Prozessmodellen genutzt werden [22]. Zunächst werden zwei Prozessmodelle P_1 und P_2 in (minimale) Prozessautomaten M_1 und M_2 transformiert (Kapitel 3.2.2 und 3.2.3). Anschließend wird die *upper bound b* berechnet: $b = |Q_1| \cdot |Q_2|$. Jetzt werden alle verschiedenen Wörter bis zur Länge $b-1$ simuliert und überprüft, ob M_1 und M_2 davon die gleiche Teilmenge akzeptieren. Dazu wird für jedes Wort kontrolliert, ob es entweder von beiden Automaten akzeptiert oder nicht akzeptiert wird. Ist dies der Fall, dann sind die beiden Prozessmodelle nach Satz 3 gleich, falls nicht, sind sie verschieden. Das gesamte Vorgehen ist in Algorithmus 2 dargestellt.

Der simulationsbasierte Ansatz soll nun anhand von zwei Beispielen illustriert werden. Dazu werden zunächst die beiden Prozessmodelle P_1 (BPMN) und P_2 (Declare) aus Abbildung 3.14

Algorithmus 2: Simulation-based algorithm

Input: Process Models P_1 and P_2
Output: True if the models are equal, otherwise False

```
1  (A, Q1, s01, δ1, F1) ← createProcessAutomaton(P1)
2  (A, Q2, s02, δ2, F2) ← createProcessAutomaton(P2)
3  n ← |Q1|
4  m ← |Q2|
5  b ← m · n
6  T ← generate all words of length ≤ b
7  for t ∈ T do
8  |  if accepts(M1, t) ≠ accepts(M2, t) then
9  |  |  return False
10 |  end
11 end
12 return True
```



precedence(A, B)	exactly$(A, 1)$
precedence(A, C)	atMost$(B, 1)$
exclusiveChoice(B, C)	atMost$(C, 1)$

Abbildung 3.14: BPMN-Prozessmodell P_1 (links) [11] und Declare-Prozessmodell P_2 (rechts)

betrachtet [11]. Die Berechnung der beiden minimalen Prozessautomaten liefert für beide Prozessmodelle das gleiche Ergebnis (Abbildung 3.15). Auf Grund der Tatsache, dass Minimalautomaten bis auf die Nummerierung der Zustände eindeutig sind [50], ist dies bei Automaten, die die gleiche Sprache erkennen, immer der Fall. Da der Automatenvergleich aber automatisiert und auch für nicht-minimierte Automaten anwendbar ist, wird das Vorgehen hier dennoch an zwei gleichen Automaten illustriert. Dafür wird zunächst die upper bound b berechnet. Beide Prozessautomaten besitzen vier Zustände. Somit beträgt $b = 4 \cdot 4 = 16$. Nun werden alle Sequenzen bis zur Länge $16 - 1 = 15$ simuliert. Anschließend wird überprüft, ob beide Automaten die gleichen Teilmengen akzeptieren. Das Ergebnis der Berechnung lautet dann

$$\{\omega \in \mathcal{L}(M_1) \mid |\omega| < 16\} = \{AB, AC\} = \{\omega \in \mathcal{L}(M_2) \mid |\omega| < 16\}$$

Das bedeutet, dass die beiden Automaten M_1 und M_2 nach Satz 3 die gleiche Sprache erkennen. Folglich sind die zu Grunde liegenden Prozessmodelle P_1 und P_2 gleich.

Als zweites Beispiel werden das BPMN-Prozessmodell P_3 aus Abbildung 3.16 sowie das Declare-Prozessmodell P_4 aus Abbildung 3.17 untersucht [11]. Die Berechnung der beiden minimalen

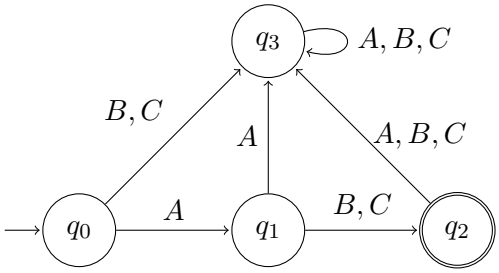


Abbildung 3.15: Prozessautomat $M_1 = M_2$ für Prozessmodelle P_1 und P_2 [11]

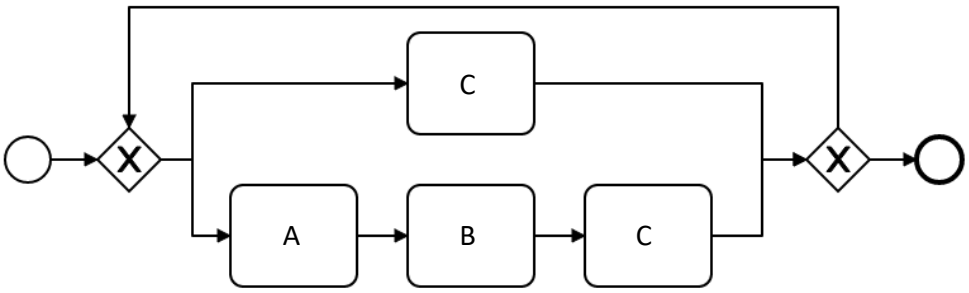
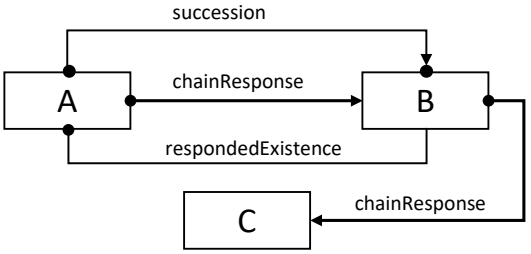


Abbildung 3.16: Prozessmodell P_3 [11]

Prozessautomaten liefert die Automaten M_3 und M_4 (Abbildung 3.18). Beide Automaten besitzen jeweils fünf Zustände. Demzufolge erhält man die *upper bound* $b = 5 \cdot 5 = 25$. Nun werden alle Sequenzen bis zur Länge $25-1 = 24$ simuliert. Danach wird überprüft, ob beide Automaten davon jeweils die gleiche Teilmenge akzeptieren. Man erkennt allerdings bereits bei Sequenzlänge null, dass M_3 das leere Wort ε nicht akzeptiert, M_4 indes schon. Folglich sind die beiden Prozessmodelle P_3 und P_4 nicht gleich.



succession(A, B)
chainResponse(A, B)
chainResponse(B, C)
respondedExistence(A, B)

Abbildung 3.17: Graphische ConDec-Repräsentation (links) und äquivalente textuelle Repräsentation (rechts) des Declare-Prozessmodells P_4

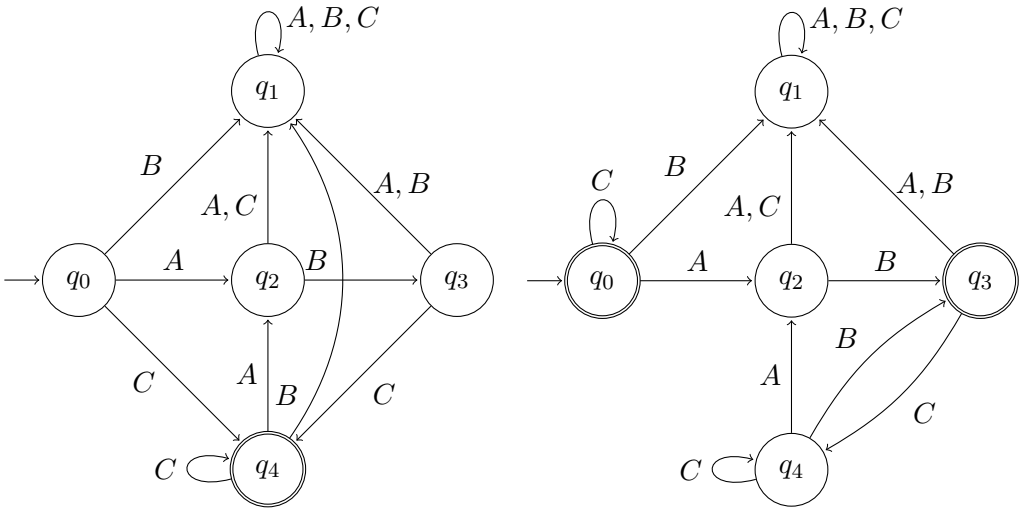


Abbildung 3.18: Prozessautomaten M_3 (links) und M_4 (rechts) für P_3 und P_4 [11]

3.2.5 Abschätzung der Upper Bound für Declare-Prozessmodelle

Dieses Kapitel beschäftigt sich mit der Beschreibung einer alternativen Methode, die es ermöglicht, zwei Declare-Prozessmodelle auf Gleichheit zu überprüfen, ohne die zugehörigen Prozessautomaten explizit zu berechnen [21]. Seien dafür $P_1 = (\mathcal{A}, \mathcal{T}_1)$ und $P_2 = (\mathcal{A}, \mathcal{T}_2)$ zwei Declare-Prozessmodelle. Des Weiteren bezeichne M_τ den Constraintautomaten (Kapitel 3.2.2) zu einem Constraint $\tau \in \mathcal{T}_i$. Dann repräsentieren die beiden (nicht minimalen) Automaten

$$M_1 := \prod_{\tau \in \mathcal{T}_1} M_\tau$$

und

$$M_2 := \prod_{\tau \in \mathcal{T}_2} M_\tau$$

die beiden Prozessmodelle P_1 und P_2. Diese Automaten unterscheiden sich vom jeweiligen Prozessautomaten nur in der Anzahl der Zustände und Übergänge, akzeptieren aber genau die gleichen Wörter. Diese beiden Automaten besitzen $c_1 := \prod_{\tau \in \mathcal{T}_1} |M_\tau|$ bzw. $c_2 := \prod_{\tau \in \mathcal{T}_2} |M_\tau|$ Zustände, wobei $|M_\tau|$ die Anzahl der Zustände des Constraintautomaten M_τ bezeichnet. Wendet man Satz 3 nun auf M_1 und M_2 an, dann erhält man als Resultat, dass diese gleich sind, falls sie die gleichen Wörter bis zur Länge $c := c_1 \cdot c_2$ akzeptieren.

Tabelle 3.3 zeigt die Anzahl der Zustände sämtlicher Constraintautomaten. Mit Hilfe dieser Werte kann der Wert c berechnet werden, ohne dass dabei die konkreten Prozessautomaten ermittelt werden müssen. Die Simulation und Überprüfung auf Akzeptanz der Wörter können

Constraint	#Zustände	**Constraint**	#Zustände
existence(A, n)	$n+1$	alternatePrecedence(A, B)	3
absence(A, n)	$n+1$	chainPrecedence(A, B)	3
succession(A, B)	3	chainSuccession(A, B)	3
init(A)	3	alternateSuccession(A, B)	3
last(A)	2	notCoExistence(A, B)	4
respondedExistence(A, B)	3	notRespondedExistence(A, B)	4
response(A, B)	2	notResponse(A, B)	3
alternateResponse(A, B)	3	notChainResponse(A, B)	3
chainResponse(A, B)	3	choice(A, B)	2
precedence(A, B)	3	exclusiveChoice(A, B)	4
coExistence(A, B)	4		

Tabelle 3.3: Anzahlen der Zustände aller Constraintautomaten [21]

response(A, B) precedence(A, B) respondedExistence(A, B)

succession(A, B) chainResponse(A, B) chainResponse(B, C)

Abbildung 3.19: Declare-Modelle M_1 und M_2

dann mit Hilfe eines Logikframeworks wie z.B. Alloy[13] erfolgen.

Im Allgemeinen ist der Wert c viel größer als die eigentliche *upper bound* b, da bei der Minimalisierung die Zustandsexplosion erheblich reduziert wird. Dies wird bei der Betrachtung der beiden Modelle M_1 und M_2 (Abbildung 3.19) deutlich. Man erhält $c_1 = 2 \cdot 3 \cdot 3 = 18$ und $c_2 = 3 \cdot 3 \cdot 3 = 27$. Somit müsste man alle Sequenzen bis zur Länge $c = c_1 \cdot c_2 - 1 = 485$ simulieren. Eine Berechnung der *upper bound* wie in Kapitel 3.2.4 liefert $b = 20$. Infolgedessen müsste man nur bis zur Länge $20 - 1 = 19$ simulieren.

3.2.6 Inklusionsbasierter Ansatz

In diesem Kapitel wird eine alternative Methode zum Vergleich zweier Prozessmodelle P_1 und P_2 vorgestellt (zweiter Schritt (*Auf Gleichheit überprüfen*) in Abbildung 3.1). Lag das Hauptaugenmerk des ersten Ansatzes, d.h. des simulationsbasierten Ansatzes (Kapitel 3.2.4), auf der Simulation aller Wörter bis zu einer gegeben Länge (*upper bound*), werden jetzt die beiden Prozessmodelle basierend auf der *symmetrischen Differenz* (Kapitel 2.2.9) der korrespondierenden Prozessautomaten verglichen. Dabei wird nach dem in Publikation [22] des Autors beschriebenen Verfahren vorgegangen.

Dazu werden zunächst wie beim simulationsbasierten Ansatz die beiden (minimalen) Prozess-

[13] `http://www.alloytools.org`, abgerufen am: 27. Dezember 2023

automaten M_1 und M_2 berechnet. Anschließend wird der Automat $M_1 \Delta M_2$ für die symmetrische Differenz der beiden Automaten berechnet. Dieser akzeptiert alle Wörter, die entweder von M_1 oder von M_2 akzeptiert werden. Folglich gilt:

$$\mathcal{L}(M_1 \Delta M_2) = \emptyset \iff \mathcal{L}(M_1) = \mathcal{L}(M_2)$$

Das bedeutet, dass die Sprache des Automaten für das symmetrische Produkts $M_1 \Delta M_2$ genau dann leer ist, wenn die Sprachen der beiden Automaten M_1 und M_2 gleich sind. Deshalb wird im nächsten Schritt überprüft, ob die Sprache des Automaten für das symmetrische Produkt leer ist. Die geschieht, indem der Automat $M_1 \Delta M_2$ minimiert wird und anschließend überprüft wird, ob der minimierte Automat einen akzeptierenden Zustand besitzt. Falls nicht, ist das symmetrische Produkt leer, und somit sind die beiden zu Grunde liegenden Prozessmodelle P_1 und P_2 gleich. Algorithmus 3 fasst die komplette Vorgehensweise zusammen. Zur Illustration

Algorithmus 3: Inclusion-based algorithm

Input: Process Models P_1 and P_2
Output: True if the models are equal, otherwise False

```
1 M_1 ← createProcessAutomaton(P_1)
2 M_2 ← createProcessAutomaton(P_2)
3 M ← M_1 Δ M_2
4 M ← minimalize(M)
5 if L(M_1 Δ M_2) = ∅ then
6 |   return True
7 else
8 |   return False
9 end
```

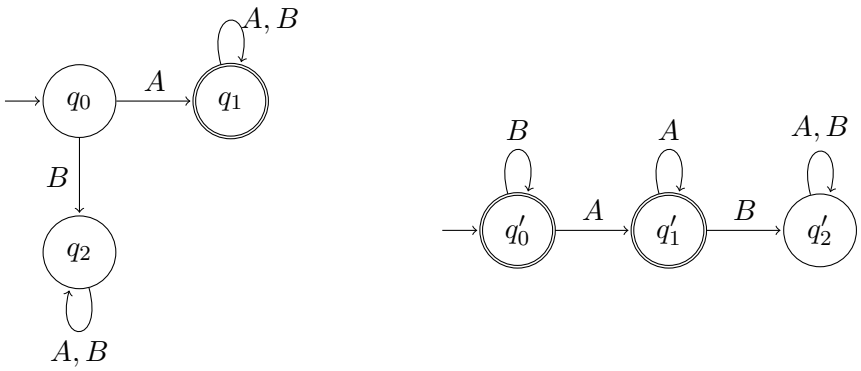


Abbildung 3.20: DEAs M_1 (links) und M_2 (rechts)

des inklusionsbasierten Ansatzes wird dieser auf die beiden DEAS M_1 und M_2 aus Kapitel

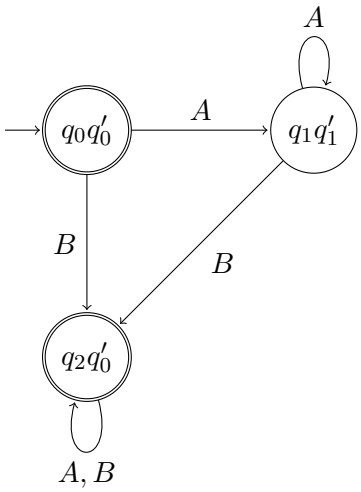


Abbildung 3.21: (Minimierter) Automat für das symmetrische Produkt $M_1 \Delta M_2$ von M_1 und M_2

2.2.1, die nochmals in Abbildung 3.20 dargestellt sind, angewandt. Abbildung 3.21 zeigt den zugehörigen Automaten $M_1 \Delta M_2$ für das symmetrische Produkt von M_1 und M_2. $M_1 \Delta M_2$ besitzt zwei akzeptierende Zustände und folglich sind die Sprachen der beiden Automaten M_1 und M_2 nach dem inklusionsbasierten Ansatz nicht gleich. Dies lässt sich manuell verifizieren, da z.B. $A \in \mathcal{L}(M_1)$, aber $A \notin \mathcal{L}(M_2)$ gilt.

3.2.7 Überprüfung auf Teilmengenbeziehungen

Dieses Kapitel widmet sich der Überprüfung von (ungleichen) Prozessmodellen auf gegenseitige Teilmengenbeziehungen (Schritt *Auf Teilmengenbeziehungen überprüfen* in Abbildung 3.1). Gerade nach Veränderungen von Modellen, z.B. nach Hinzufügen von zusätzlichen Constraints oder Alternativpfaden, stellt sich die Frage, ob das neu entstandene Modell „restriktiver“ als das ursprüngliche Modell ist oder ob es „flexibler“ geworden ist, d.h. ob es mehr Ausführungspfade besitzt als das ursprüngliche Modell. Diese Frage kann mit der in diesem Kapitel beschriebenen Methode beantwortet werden. Dazu gilt es zunächst zu klären, was unter *Teilmengenbeziehungen von Prozessmodellen* zu verstehen ist:

Definition 20. *Seien P_1 und P_2 zwei Prozessmodelle und M_1 und M_2 die korrespondierenden Prozessautomaten. P_1 ist **Teilmenge** von P_2 ($P_1 \subseteq P_2$), falls $\mathcal{L}(M_1) \subseteq \mathcal{L}(M_2)$.*

Das bedeutet, dass ein Prozessmodell P_1 in einem anderen Prozessmodell P_2 enthalten ist, falls alle akzeptierten Sequenzen von P_1 auch akzeptierte Sequenzen von P_2 sind. Zur Überprüfung der Teilmengenbeziehungen werden die sogenannten *Differenzautomaten* $M_1 \setminus M_2$ und $M_2 \setminus M_1$ (Kapitel 2.2.8) der entsprechenden Prozessautomaten M_1 und M_2 berechnet [22].

Anschließend wird überprüft, ob diese leer sind, d.h. die leere Sprache erkennen. Falls einer der Differenzautomaten $M_i \setminus M_j$ leer ist, folgt, dass $\mathcal{L}(M_i) \subseteq \mathcal{L}(M_j)$ (Kapitel 2.2.8) und somit $P_i \subseteq P_j$. Das gesamte Vorgehen ist in Algorithmus 4 dargestellt.

Algorithmus 4: mutualContainment

Input: (unequal) Process Models P_1 and P_2
Output: Mutual containment relationship, otherwise False

```
1  M_1 ← createProcessAutomaton(P_1)
2  M_2 ← createProcessAutomaton(P_2)
3  D_1 ← M_1 \ M_2
4  D_2 ← M_2 \ M_1
5  if L(D_1) = ∅ then
6  |  return P_1 subset of P_2
7  else if L(D_2) = ∅ then
8  |  return P_2 subset of P_1
9  else
10 |  return false
11 end
```

Zur Illustration des Vorgehens werden wiederum die beiden DEAs M_1 und M_2 aus Abbildung 3.22 betrachtet. Es wird in der Folge überprüft, ob die Sprache von M_2 eine Teilmenge der Sprache von M_1 ist. Dafür wird der Komplementautomat ${M_1}^C$ von M_1 benötigt, der in Abbildung 3.22 dargestellt ist. Nun kann der Differenzautomat $M_2 \setminus M_1 = M_2 \times {M_1}^C$ berechnet werden. Dieser ist in Abbildung 3.23 zu sehen. $M_2 \setminus M_1 = M_2 \times {M_1}^C$ besitzt offensichtlich zwei akzeptierende Zustände, und demnach ist die Sprache von M_2 keine Teilmenge der Sprache von M_1. Dies ist auch manuell ersichtlich, da beispielsweise $\varepsilon \in \mathcal{L}(M_2)$, aber $\varepsilon \notin \mathcal{L}(M_1)$ gilt.

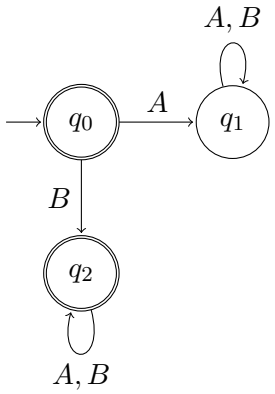


Abbildung 3.22: Komplementautomat ${M_1}^C$ von M_1

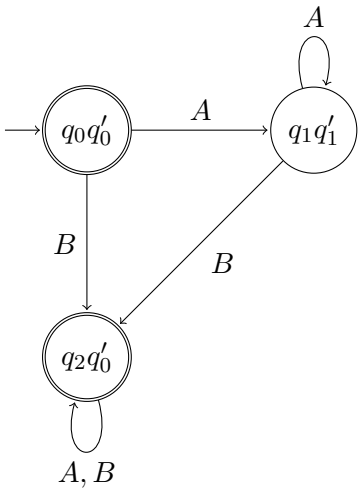


Abbildung 3.23: Differenzautomat $M_2 \setminus M_1 = M_2 \times {M_1}^C$

3.2.8 Berechnung von Gemeinsamkeiten

Für den Fall, dass zwei (verschiedene) Prozessmodelle nicht in einer Teilmengenbeziehung stehen, stellt sich die Frage, worin die Gemeinsamkeiten und Unterschiede der Modelle bestehen. Dazu wird im Folgenden eine Methode zur Berechnung der Gemeinsamkeiten präsentiert (Schritt *Produktautomat berechnen* in Abbildung 3.1) [22]. Unterkapitel 3.2.9 behandelt hingegen die Berechnung der Unterschiede.

Die Gemeinsamkeiten zweier Prozessmodelle P_1 und P_2 werden mit Hilfe des *Produktautomaten* (Kapitel 2.2.6) $M_1 \times M_2$ der beiden entsprechenden Prozessautomaten M_1 und M_2 berechnet. Die Sprache des Produktautomaten besteht aus genau den Sequenzen, die von beiden Prozessmodellen akzeptiert werden:

$$\mathcal{L}(M_1 \times M_2) = \{t \mid t \text{ erfüllt } P_1\} \cap \{t \mid t \text{ erfüllt } P_2\}$$

Algorithmus 5 zeigt das Vorgehen zur Berechnung der Gemeinsamkeiten.

Algorithmus 5: calculateIntersection

Input: (unequal) Process Models P_1 and P_2
Output: Automaton for intersection of P_1 and P_2

```
1 M1 ← createProcessAutomaton(P1)
2 M2 ← createProcessAutomaton(P2)
3 M ← M1 × M2
4 return M
```

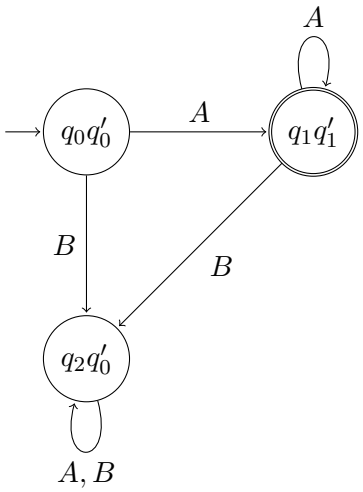


Abbildung 3.24: Produktautomat $M = M_1 \times M_2$

Aus dem Produktautomaten kann ein äquivalenter regulärer Ausdruck (Kapitel 2.2.2) abgeleitet werden. Dieser kann von einer Domänenexpertin oder einem Domänenexperten interpretiert werden. Alternativ kann man auch Beispielsequenzen simulieren, um so ein Gefühl für die Gemeinsamkeiten zu bekommen.

Abschließend werden wieder die beiden DEAs M_1 und M_2 aus Abbildung 3.20 und deren zugehöriger Produktautomat $M = M_1 \times M_2$ (Abbildung 3.24) betrachtet, der die Gemeinsamkeiten von M_1 und M_2 beschreibt. Man sieht, dass M genau alle Wörter akzeptiert, die ausschließlich das Symbol A beinhalten, d.h. $\mathcal{L}(M) = \{A, AA, AAA, ...\}$. Ein äquivalenter regulärer Ausdruck wäre demnach $a = A(A)^* = AA^*$.

3.2.9 Berechnung von Unterschieden

In Kapitel 3.2.8 werden die Gemeinsamkeiten von verschiedenen, nicht ineinander enthaltenen Prozessmodellen P_1 und P_2 betrachtet. Jetzt sollen deren Unterschiede berechnet werden (Schritt *Differenzautomaten berechnen* in Abbildung 3.1) [22]. Dafür werden die beiden Differenzautomaten (Kapitel 2.2.8) $M_1 \setminus M_2$ und $M_2 \setminus M_1$ der beiden entsprechenden Prozessautomaten M_1 und M_2 berechnet. Diese akzeptieren jeweils alle Sequenzen, die von einem Modell akzeptiert, vom anderen Modell aber nicht akzeptiert werden und beschreiben somit den Unterschied zwischen den beiden Prozessmodellen P_1 und P_2:

$$\mathcal{L}(M_1 \setminus M_2) = \{t \mid t \text{ erfüllt } P_1\} \cap \{t \mid t \text{ erfüllt } P_2 \text{ nicht}\}$$

Algorithmus 6 zeigt das Vorgehen zur Berechnung der Unterschiede.

Algorithmus 6: calculateDifferences

Input: (unequal) Process Models P_1 and P_2
Output: Automatons for differences of P_1 and P_2

```
1 M_1 ← createProcessAutomaton(P_1)
2 M_2 ← createProcessAutomaton(P_2)
3 D_1 ← M_1 \ M_2
4 D_2 ← M_2 \ M_1
5 return D_1, D_2
```

Aus den beiden Differenzautomaten lässt sich jeweils ein äquivalenter regulärer Ausdruck ableiten (siehe Kapitel 2.2.2). Dieser reguläre Ausdruck kann von Fachleuten auf dem Gebiet interpretiert werden. Eine alternative Herangehensweise besteht darin, Beispielsequenzen zu simulieren, um so das Bewusstsein für die konkreten Unterschiede zwischen den Prozessmodellen zu schärfen.

Zur Erklärung des Vorgehens wird abermals das Verhalten der beiden DEAs M_1 und M_2 aus Abbildung 3.20 untersucht. Der korrespondierende Differenzautomat $M_2 \setminus M_1$ ist bereits in Kapitel 3.2.7 in Abbildung 3.23 abgebildet. Man erkennt, dass dieser die leere Sequenz ε und alle Wörter, die mit B beginnen, akzeptiert, d.h. $\mathcal{L}(M_2 \setminus M_1) = \{\varepsilon, B, BA, BB, ...\}$. Ein äquivalenter regulärer Ausdruck wäre demzufolge $a = \varepsilon \mid (B(A \mid B)^*)$.

3.3 Vergleich weiterer Perspektiven

In Kapitel 3.2 werden zwei verschiedene Ansätze vorgestellt, die es ermöglichen, Prozessmodelle auf Gleichheit bezüglich der Kontrollflussperspektive zu überprüfen: der simulationsbasierte Ansatz (Kapitel 3.2.4) und der inklusionsbasierte Ansatz (Kapitel 3.2.6). Darüber hinaus ist auch interessant, Prozessmodelle nicht nur auf Gleichheit bezüglich des Kontrollflusses, sondern auch hinsichtlich weiterer Aspekte, z.B. am Prozess beteiligte Personen, Werkzeuge, Programme oder Datenobjekte, zu behandeln. Diesen Aspekten wird sich in den kommenden Unterkapiteln gewidmet, indem Vergleichsmethoden bezüglich der organisatorischen Perspektive (Kapitel 3.3.1), der operationalen Perspektive (Kapitel 3.3.2) sowie der Daten- und Datenflussperspektive (Kapitel 3.3.3) eingeführt werden. Dabei wird für alle Perspektiven davon ausgegangen, dass die beiden zu vergleichenden Prozessmodelle gleich bezüglich des Kontrollflusses sind. Des Weiteren wird vorausgesetzt, dass externe Verwaltungsstrukturen (z.B. Organisationsmodelle) bereits beim Vergleich der funktionalen Perspektive auf Äquivalenz überprüft und gegebenenfalls angepasst wurden.

Generell sind die im Folgenden vorgestellten Methoden zum Vergleich weiterer Perspektiven vollständig unabhängig voneinander anwendbar. Das bedeutet, dass je nach spezifischem An-

wendungsfall die erforderlichen Perspektiven ausgewählt werden können. Insbesondere ist die Reihenfolge des Vergleichs weiterer Perspektiven irrelevant und kann nach Belieben gewählt werden.

3.3.1 Vergleich der organisatorischen Perspektive

In diesem Kapitel werden Prozessmodelle, die strukturgleich sind, d.h. gleich sind bezüglich der Kontrollflussperspektive, genauer hinsichtlich der organisatorischen Perspektive (Kapitel 2.1.3) untersucht. Dabei ist es in der Literatur üblich, solche Untersuchungen unabhängig von spezifischen Modellierungssprachen durchzuführen [56, 57]. Daher werden lediglich Bedingungen betrachtet, die in einer beliebigen Logik formuliert sind (z.B. Aussagenlogik). Diese Bedingungen stellen Einschränkungen an die organisatorische Perspektive bezüglich der Aktivitäten des Prozesses dar. Solch eine Einschränkung kann beispielsweise sein: „Aktivität A muss immer von Max Mustermann ausgeführt werden“ oder „Die ausführende Person einer bestimmten Aktivität muss mindestens 18 Jahre alt sein“. Dies sind Bedingungen, die typischerweise in jedem gängigen Prozessmanagementsystem umgesetzt werden können.

Es wird grundsätzlich davon ausgegangen, dass die organisatorische Perspektive zur Deklaration der Ausführenden einer gewissen Aktivität A in einem Prozess durch eine Formel $C(A)$ in einer beliebigen Logik ausgedrückt ist. Dann lässt sich $C(A)$ äquivalent in konjunktive Normalform (KNF) zerlegen, sodass

$$C(A) = C_1(A) \wedge \cdots \wedge C_n(A),$$

wobei die Terme $C_i(A)$ paarweise verschieden sind und jeder Term $C_i(A)$ jeweils nur noch eine elementare Bedingung beinhaltet [58]. Unter einer „elementaren Bedingung“ versteht man eine Bedingung, die lediglich an eine Variable geknüpft ist, z.B: „Alter > 18“ oder „Geschlecht = 'm'“. Die konjunktive Normalform wird berechnet, da in dieser standardisierten Form die elementaren Bedingungen durch ein logisches Und ($\wedge$) verknüpft sind und somit die komplette Formel gut zu überblicken ist [58]. Unter der Annahme, dass alle Zuweisungen in KNF vorliegen, wird zum Vergleich der organisatorischen Perspektive prinzipiell nach folgendem Verfahren vorgegangen:

1. Bestimme relevante Domänen $D_i(A)$ für alle Teilformeln $C_i(A)$ und deren Kardinalitäten $d_i(A) := \#D_i(A)$ für alle Aktivitäten A.
2. Berechne die Gesamtzahl an verschiedenen Ausprägungen für A: $D(A) := \prod_{i=1}^{n} d_i(A)$
3. Bestimme die Menge der für $C(A)$ relevanten Domänen der zu vergleichenden Prozesse.
4. Berechne Ähnlichkeiten für jede Aktivität.
5. Berechne Ähnlichkeit für Prozesse als Mittelwert der Ähnlichkeiten der Aktivitäten.

Im Folgenden wird das oben beschriebene Verfahren Schritt für Schritt erklärt. Dafür werden konjunktive Normalformen genauer untersucht. Sei dazu $C(A) = C_1(A) \wedge \cdots \wedge C_n(A)$ eine Formel in konjunktiver Normalform wie oben beschrieben. Dann bezeichne $D_i(A)$ die Menge der verschiedenen relevanten Domänen für $C_i(A)$ und $d_i(A) := \#D_i(A)$ die zugehörige Kardinalität (Schritt 1). Darunter versteht man die Anzahl der verschieden Ausprägungen für $C_i(A)$, die jeweils zu einem anderen Ergebnis führen. Beispielsweise besitzt die Formel „Alter > 18“ zwei verschiedene relevante Ausprägungen: Entweder das Alter ist ≤ 18 (somit wäre das Ergebnis dieser Formel *false*, oder das Alter ist < 18 (wodurch das Ergebnis dieser Formel *true* wäre). Somit gibt es für den Fall, dass der Datentyp von Alter als eine natürliche Zahl definiert ist, die beiden Domänen $[0, 1, ..., 18]$ und $[19, 20, ...]$. Dieses Verfahren angewandt auf alle elementaren Formeln $C_i(A)$ liefert eine Gesamtzahl $D(A)$ an verschiedenen und relevanten Ausprägungen (Schritt 2):

$$D(A) := \prod_{i=1}^{n} d_i(A)$$

Nun lässt sich für ein Prozessmodell P und eine Aktivität A die Anzahl der relevanten Domänen für A definieren (Schritt 3):

Definition 21. *Sei P ein Prozessmodell, A eine Aktivität von P und $C(A) := C_1(A) \wedge \cdots \wedge C_n(A)$ eine Formel bezüglich A in konjunktiver Normalform. Dann heißt*

$$D_i(A, P) := \{d \in D_i(A) \mid d \text{ erfüllt } C_i(A)\} \subseteq D_i(A)$$

*die **Menge der für $C_i(A)$ relevanten Domänen von P.***
Weiter bezeichnet

$$D(A, P) := \prod_{i=1}^{n} D_i(A, P)$$

*die **Menge der für $C(A)$ relevanten Domänen von P.***

Obige Definition wird nun anhand eines kleinen Beispiels erläutert. Dazu wird die folgende Formel $C(A)$ in konjunktiver Normalform betrachtet, die einer Aktivität A eines Prozesses P die organisatorische Perspektive zuweist:

$$C(A) = \underbrace{(\text{Alter } > 18)}_{C_1(A)} \wedge \underbrace{(\text{ Geschlecht } = \text{'}m\text{'})}_{C_2(A)}$$

Damit gibt es, wie oben beschrieben, sowohl für das Alter als auch für das Geschlecht zwei verschiedene Domänen (d.h. $\#D_i(A) = 2$ für $i = 1, 2$). Die beiden Domänen für das Alter sind $[0, 1, ..., 18]$ und $[19, 20, ...]$, die für das Geschlecht $\{m\}$ und $\{w, d\}$. Daraus folgt eine Gesamtzahl an relevanten Ausprägungen $D(A) = 2 \cdot 2 = 4$, nämlich:

- Alter $\leq 18 \wedge$ Geschlecht = 'm'
- Alter $> 18 \wedge$ Geschlecht = 'm'
- Alter $\leq 18 \wedge$ Geschlecht $\in \{w, d\}$
- Alter $> 18 \wedge$ Geschlecht $\in \{w, d\}$

Sowohl für das Alter als auch für das Geschlecht ist durch obige Formel genau eine Domäne erlaubt, d.h. $d_i(A, P) = 1$ für $i = 1, 2$. Somit ist die Anzahl der für $C(A)$ relevanten Domänen $1 \cdot 1 = 1$. Der zweite Punkt in obiger Aufzählung repräsentiert diese einzige zulässige Konstellation an verschiedenen Ausprägungen.

Bemerkung. *Beim Betrachten eines einzelnen Prozessmodells beträgt die Anzahl der relevanten Domänen für eine Formel immer* 1*, da es für jede Teilformel der konjunktiven Normalform jeweils nur zwei relevante Domänen (nämlich eine, die die Teilformel erfüllt und eine, die sie nicht erfüllt) und somit eine zulässige Domäne gibt. Im Fortgang der Arbeit werden aber zwei verschiedene Prozessmodelle miteinander verglichen. Deshalb gibt es in der Regel auch mehr als zwei verschiedene relevante Ausprägungen.*

Nun werden die oben eingeführten und definierten Begrifflichkeiten zum Vergleich der organisatorischen Perspektive von zwei verschiedenen Prozessmodellen verwendet. Seien dazu P_1 und P_2 Prozessmodelle, A eine gemeinsame Aktivität und $C(A) = C_1(A) \wedge \cdots \wedge C_n(A)$ und $C'(A) = C'_1(A) \wedge \cdots \wedge C'_m(A)$ jeweils die zu Aktivität A beschreibende Formel für die organisatorische Perspektive in konjunktiver Normalform in den jeweiligen Modellen. Weiter sei $d_i(A)$ die Anzahl der relevanten Domänen für $C(A)$ und $C'(A)$. Dann wird die Ähnlichkeit $SimOrg_{P_1,P_2}(A)$ bezüglich der organisatorischen Perspektive von P_1 und P_2 bezüglich der Aktivität A folgendermaßen definiert (Schritt 4):

$$SimOrg_{P_1,P_2}(A) := \frac{\#\left(D(A, P_1) \cap D(A, P_2)\right)}{\#D(A, P_1) + \#D(A, P_2)}$$

Zur Illustration dieser Berechnung wird ein Beispiel bestehend aus zwei Prozessen P_1 und P_2 mit einer gemeinsamen Aktivität A betrachtet. Die organisatorische Perspektive ist dabei durch die beiden folgenden Formeln $C(A)$ und $C'(A)$ gegeben:

$$C(A) = \underbrace{(\text{Alter} > 60\ )}_{C_1(A)} \wedge \underbrace{(\text{Geschlecht} = \text{'}m\text{'})}_{C_2(A)} \quad \Big| \quad C'(A) = \underbrace{\text{Alter} > 59}_{C'_1(A)}$$

Das Aufstellen der für beide Prozessmodelle relevanten Domänen liefert die obigen sechs (d.h. $d_i(A) = 6$) verschiedenen Domänen. Die Formel $C(A)$ wird lediglich von Domäne (i)

(i) Alter $> 60 \ \wedge$ Geschlecht = 'm'	(ii) Alter $> 60 \ \wedge$ Geschlecht $\neq$ 'm'
(iii) $59 <$ Alter $\leq 60 \ \wedge$ Geschlecht = 'm'	(iv) $59 <$ Alter $\leq 60 \ \wedge$ Geschlecht $\neq$ 'm'
(v) Alter $\leq 59 \ \wedge$ Geschlecht = 'm'	(vi) Alter $\leq 59 \ \wedge$ Geschlecht $\neq$ 'm'

erfüllt. Daher gilt $\#D(A, P_1) = 1$. Formel $C'(A)$ wird von den Domänen $(i), (ii), (iii)$ und (iv) erfüllt, d.h. $\#D(A, P_2) = 4$. Da Domäne (i) sowohl $C(A)$ als auch $C'(A)$ erfüllt, gilt: $\#\left(D(A, P_1) \cap D(A, P_2)\right) = 1$. Somit folgt für die Ähnlichkeit bezüglich der organisatorischen Perspektive:

$$SimOrg_{P_1,P_2}(A) = \frac{1}{5} = 20\%$$

Zur Illustration der praktischen Anwendbarkeit des entwickelten Ansatzes wird dieser auf drei der gängigsten *Workflow Ressource Patterns* [59] angewendet.

Als erstes Pattern wird das Pattern *Direct Allocation* untersucht. Dabei wird einer Aktivität direkt ein Element der organisatorischen Perspektive zugeordnet. Beispielsweise könnte die Zuweisung für eine Aktivität A der Form $C(A) = \{\text{exec}(A) = \text{'Alice'}\}$ sein. Demzufolge gäbe es in diesem Fall die beiden verschiedenen Domänen $D_1 = \{\text{Alice}\}$ und $D_2 = S \setminus \{\text{Alice}\}$, wobei S die Menge aller Elemente der organisatorischen Perspektive bezeichnet.

Das zweite Pattern ist das Pattern *Role-Based Allocation*. Dabei wird einer Aktivität eine Gruppe oder Rolle (engl. *role*) zugeordnet. Hier könnte eine Zuweisung für eine Aktivität A die Form $C(A) = \{\text{exec}(A) \in \text{Studierender}\}$ besitzen. Diese Zuweisung kann prinzipiell genauso aufgelöst werden wie beim Patter *Direct Allocation*. Die Domäne der Studierenden ist ebenfalls eine Untermenge aller Elemente des Organisationsmodells. Daher können hier auch zwei Domänen aufgestellt werden, nämlich $D_1 = \text{Studierender}$ und $D_2 = S \setminus \text{Studierender}$, wobei S die Menge aller Elemente der organisatorischen Perspektive bezeichnet.

Als drittes Pattern wird das Pattern *Separation of Duties* betrachtet. Dieses wird in dieser Arbeit mit dem „Vieraugenprinzip“ gleichgesetzt. Das bedeutet, dass zwei Aktivitäten A und B nicht von der gleichen Person erledigt werden dürfen. So könnten die Zuweisungen etwa folgendermaßen aussehen: $C(A) = \{\text{exec}(A) = \text{'Alice'}\}$ und $C(B) = \{\text{exec}(B) \neq \text{exec}(A)\}$. Der Vergleich für Aktivität A erfolgt nach dem Standardvorgehen, da es sich um eine direkte Zuweisung (*Direct Allocation*) handelt. Für Aktivität B werden dann die beiden Domänen $D_1 = \text{exec}(A)$ und $D_2 = S \setminus \text{exec}(A) = S \setminus \{\text{Alice}\}$ eingeführt, wobei S die Menge aller Elemente der organisatorischen Perspektive bezeichnet. Somit kann auch das Pattern *Separation of Duties*, das sich auf zwei verschiedene Aktivitäten bezieht, umgesetzt werden.

Die Übertragbarkeit des entwickelten Verfahrens, welches beispielhaft anhand von drei Pattern veranschaulicht wurde, ist ebenfalls auf weitere Pattern gegeben, z.B. *Case Handling* oder *Organisational Allocation* [59].

Bisher wird ein Ähnlichkeitswert lediglich für Aktivitäten berechnet, die paarweise verglichen

werden. Abschließend wird der Ansatz auf komplette Prozessmodelle erweitert (Schritt 5). Seien dafür P_1 und P_2 zwei Prozessmodelle mit einer gemeinsamen (endlichen) Menge an Aktivitäten $\mathcal{A}$. Dann wird die Ähnlichkeit von P_1 und P_2 bezüglich der organisatorischen Perspektive folgendermaßen definiert:

$$SimOrg(P_1, P_2) := \frac{\sum_{A \in \mathcal{A}} SimOrg_{P_1,P_2}(A)}{\#\mathcal{A}}$$

Beispielhaft werden zwei Prozesse P_1 und P_2 mit drei gemeinsamen Aktivitäten A, B und C betrachtet. Diese Aktivitäten besitzen bezüglich der organisatorischen Perspektive folgende Zuweisungen:

Aktivität	P_1	P_2
A	Alice	Bob
B	exec(A)	exec(A)
C	¬exec(A)	exec(B)

Vergleicht man die Zuweisungen zu Aktivität A, erhält man (unter der Annahme, dass Alice und Bob keine korrespondierenden Elemente in den jeweiligen Organisationsmodellen sind) die drei Domänen {Alice}, {Bob} und $S \setminus$ {Alice, Bob}, wobei S die Menge aller Elemente der organisatorischen Perspektive bezeichnet. Demnach ergibt sich die Ähnlichkeit $SimOrg_{P_1,P_2}(A) = 0$. Analog erhält man $SimOrg_{P_1,P_2}(B) = 0$, da sowohl in P_1 als auch in P_2 Aktivität B der oder die Ausführende von A zugewiesen ist. Für Aktivität C erhält man die folgenden vier Domänen: $D_1 = S \setminus \{\text{exec}(A)\} = S \setminus \{\text{Alice}\}$, $D_2 = \{\text{Alice}\}$, $D_3 = \{\text{exec}(B) = \text{exec}(A) = \text{Bob}\}]$ und $D_4 = S \setminus \{\text{exec}(B)\} = S \setminus \{\text{Bob}\}$. Lediglich Domäne D_4 erfüllt die beiden Zuweisungen von P_1 und P_2. Demnach ergibt sich für Aktivität C eine Ähnlichkeit von $SimOrg_{P_1,P_2}(C) = \frac{1}{4} = 0,25$.

$$SimOrg(P_1, P_2) = \frac{SimOrg_{P_1,P_2}(A) + SimOrg_{P_1,P_2}(B) + SimOrg_{P_1,P_2}(C)}{3} =$$

$$= \frac{0 + 0 + 0,25}{3} = \frac{0,25}{3} = 0,08\overline{3} = 8,\overline{3}\%$$

Bei einer Ähnlichkeit von $8,\overline{3}\%$ lässt sich vermuten, dass die beiden Prozesse P_1 und P_2 ziemlich unähnlich bezüglich der organisatorischen Perspektive sind. Somit wäre eine Fusion bzw. eine direkte Anpassung der beiden Prozesse aneinander zunächst einmal nur mit relativ großem Aufwand umsetzbar. Allerdings könnte eine Domänenexpertin oder ein Domänenexperte erkennen, dass der gravierende Unterschied zwischen den beiden Prozessen in der Zuweisung von Aktivität A liegt. Eine Anpassung eines der beiden Prozesse hinsichtlich der ausführenden Person von Aktivität A (z.B. durch das Einführen einer gemeinsamen Rolle für Alice und Bob) hätte zur Folge, dass $SimOrg_{P_1,P_2}(A) = SimOrg_{P_1,P_2}(B) = 1$ und $SimOrg_{P_1,P_2}(C) = 0$,

was eine Gesamtähnlichkeit von $SimOrg(P_1, P_2) = \frac{2}{3} = 66,\overline{6}\%$ liefert. Hier wäre der einzige verbleibende Unterschied zwischen den beiden Modellen, dass die Zuweisungen zu Aktivität C genau gegensätzlich sind ($\neg$exec(A) vs. exec(B) =exec(A)).

Für detaillierte Interpretationen der in diesem Abschnitt vorgestellten Verfahren und Berechnungen wird auf Kapitel 3.3.4 verwiesen.

3.3.2 Vergleich der operationalen Perspektive

Nachdem Prozessmodelle bislang auf Gleichheiten bezüglich der funktionalen Perspektive (Kapitel 3.1), der Kontrollflussperspektive (Kapitel 3.2) sowie der organisatorischen Perspektive (Kapitel 3.3.1) verglichen wurden, wird nun eine Methode zum Vergleich der operationalen Perspektive (Kapitel 2.1.3) beschrieben.

Die operationale Perspektive wird in einem Prozessmodell ebenso wie die organisatorische Perspektive den Aktivitäten zugewiesen. Daher können alle in Kapitel 3.3.1 eingeführten Definitionen und Methoden für die organisatorische Perspektive gleichermaßen für die operationale Perspektive angewendet werden. Die beiden Perspektiven unterscheiden sich lediglich im zu Grunde liegenden Organisationsmodell [14]: Hier werden anstatt Organisationen (z.B. Personen, Rollen, Firmen) eben Elemente der operationalen Perspektive (z.B. Werkzeuge, Programme) verwaltet. Daher kann die Strukturierung des Ansatzes aus Abschnitt 3.3.1 gleichermaßen angewendet werden, da die Hintergrundmodelle strukturgleich sind.

Der komplette Ansatz aus Kapitel 3.3.1 kann demnach für die operationale Perspektive übernommen werden. Die Ähnlichkeit von zwei Prozessmodellen P_1 und P_2 bezüglich einer Aktivität A bezüglich der operationalen Perspektive wird folgendermaßen definiert:

$$SimOp_{P_1,P_2}(A) := \frac{\#\left(D(A,P_1) \cap D(A,P_2)\right)}{\#D(A,P_1) + \#D(A,P_2)}$$

Dabei sind die Domänen $D(A, P_i)$ für $i = 1, 2$ wie in Kapitel 3.3.1 definiert. Nun kann obige Definition noch auf komplette Prozessmodelle P_1 und P_2 über einer Menge an Aktivitäten $\mathcal{A}$ erweitert werden:

$$SimOp(P_1, P_2) := \frac{\sum_{A \in \mathcal{A}} SimOp_{P_1,P_2}(A)}{\#\mathcal{A}}$$

Beispielhaft werden zwei Prozesse P_1 und P_2 mit drei gemeinsamen Aktivitäten A, B und C betrachtet. Diese Aktivitäten besitzen bezüglich der operationalen Perspektive folgende Gleichheiten: $SimOp_{P_1,P_2}(A) = 0,3, SimOp_{P_1,P_2}(B) = 0,5$ und $SimOp_{P_1,P_2}(C) = 0,6$. Demzufolge ergibt sich für die Gleichheit von P_1 und P_2 bezüglich der organisatorischen Perspektive:

$$SimOp(P_1, P_2) = \frac{SimOp_{P_1,P_2}(A) + SimOp_{P_1,P_2}(B) + SimOp_{P_1,P_2}(C)}{3} =$$

$$= \frac{0,3 + 0,5 + 0,6}{3} = \frac{1,4}{3} = 0,4\overline{6} \approx 47\%$$

Eine Ähnlichkeit von 47% lässt vermuten, dass die beiden Prozesse P_1 und P_2 ziemlich ungleich bezüglich der operationalen Perspektive sind. Das hätte zur Folge, dass für eine Fusion bzw. eine Anpassung der beiden Prozesse aneinander ein relativ großer Aufwand notwendig wäre. Zur ausführlichen Interpretation der in diesem Abschnitt präsentierten Methoden und Berechnungen wird auf Kapitel 3.3.4 verwiesen.

3.3.3 Vergleich der Daten- und Datenflussperspektive

Als letzte Perspektive wird die Daten- und Datenflussperspektive (Kapitel 2.1.3) untersucht. Hierfür wird in diesem Kapitel eine Vergleichsmethode entwickelt, die es ermöglicht, Prozessmodelle bezüglich dieser Perspektive zu vergleichen.

Die Daten- und Datenflussperspektive beschreibt im Prozess auftretende Datenobjekte und Dokumente. Diese werden jeweils den Aktivitäten als In- und Output zugewiesen. Somit können Datenobjekte durch das Ausführen von Aktivitäten bearbeitet oder erstellt werden. Für die entwickelte Vergleichsmethode werden zunächst nur einzelne Aktivitäten und deren zugewiesene Datenobjekte betrachtet. Seien dafür P_1 und P_2 Prozessmodelle, A eine gemeinsame Aktivität von P_1 und P_2 sowie D die Menge der Datenobjekte, die in P_1 und P_2 verwendet werden. Weiter seien $I_{P_i}(A) \subseteq D$ die Menge der Inputs von A in P_i für $i = 1, 2$. Dann ist die Ähnlichkeit $SimInput_{P_1,P_2}(A)$ von P_1 und P_2 bezüglich A bezüglich der Inputs folgendermaßen definiert:

$$SimInput_{P_1,P_2}(A) := \frac{\#\left(I_{P_1}(A) \cap I_{P_2}(A)\right)}{\#I_{P_1}(A) + \#I_{P_2}(A) - \#\left(I_{P_1}(A) \cap I_{P_2}(A)\right)} \in [0,1]$$

Das bedeutet, dass die Ähnlichkeit über das Verhältnis der gemeinsamen Inputs zu allen Inputs definiert wird. Analog wird die Ähnlichkeit der Outputs definiert:

$$SimOutput_{P_1,P_2}(A) := \frac{\#\left(O_{P_1}(A) \cap O_{P_2}(A)\right)}{\#O_{P_1}(A) + \#O_{P_2}(A) - \#\left(O_{P_1}(A) \cap O_{P_2}(A)\right)} \in [0,1]$$

Nun kann die Ähnlichkeit von P_1 und P_2 bezüglich A bezüglich der Daten- und Datenflussperspektive definiert werden. Dazu werden die Ähnlichkeiten für Inputs und Outputs gemittelt:

$$SimData_{P_1,P_2}(A) := \frac{SimInput_{P_1,P_2}(A) + SimOutput_{P_1,P_2}(A)}{2} \in [0,1]$$

Zur Illustration der Berechnung der Ähnlichkeit wird ein Beispiel zweier Prozessmodelle P_1 und P_2 mit einer gemeinsamen Aktivität betrachtet. Angenommen, A besitzt in P_1 die Inputs $I_{P_1}(A) = \{D_1, D_2, D_3\}$ und Outputs $O_{P_1}(A) = \{D_1, D_3\}$. Zudem besitzt A in P_2 die Inputs

$I_{P_2}(A) = \{D_1, D_3\}$ und Outputs $O_{P_2}(A) = \{D_1, D_2\}$. Daraus ergibt sich für die Gleichheit bezüglich der Daten- und Datenflussperspektive:

$$SimData_{P_1,P_2}(A) := \frac{SimInput_{P_1,P_2}(A) + SimOutput_{P_1,P_2}(A)}{2} =$$

$$= \frac{\overbrace{\frac{2}{3+2-2}}^{SimInput_{P_1,P_2}(A)} + \overbrace{\frac{1}{2+2-1}}^{SimOutput_{P_1,P_2}(A)}}{2} = \frac{\frac{2}{3} + \frac{1}{3}}{2} = \frac{1}{2} = 50\%$$

Nun wird die Ähnlichkeit bezüglich der Daten- und Datenflussperspektive auf komplette Prozessmodelle P_1 und P_2 über einer Menge an Aktivitäten $\mathcal{A}$ erweitert:

$$SimData(P_1, P_2) := \frac{\sum_{A \in \mathcal{A}} SimData_{P_1,P_2}(A)}{\#\mathcal{A}}$$

Abschließend werden zwei Prozessmodelle P_1 und P_2 über der Menge an Aktivitäten $\mathcal{A} = \{A, B\}$ betrachtet. Die Aktivitäten besitzen dabei die folgenden In- und Outputs:

	A	B
I_{P_1}	$\{D_1, D_2, D_3\}$	$\{D_1, D_3\}$
I_{P_2}	$\{D_1, D_3\}$	$\{D_1, D_2, D_3\}$
O_{P_1}	$\{D_1, D_3\}$	$\{D_2, D_3\}$
O_{P_2}	$\{D_1, D_2\}$	$\{D_1\}$

Die Ähnlichkeit von Aktivität A bezüglich der Daten- und Datenflussperspektive wurde bereits oben berechnet ($SimData_{P_1,P_2}(A) = 0,5$). Nun wird die Ähnlichkeit von Aktivität B bestimmt:

$$SimData_{P_1,P_2}(B) = \frac{SimInput_{P_1,P_2}(B) + SimOutput_{P_1,P_2}(B)}{2} =$$

$$= \frac{\frac{2}{2+3-2} + \frac{0}{2+1-0}}{2} = \frac{\frac{2}{3} + 0}{2} = \frac{\frac{2}{3}}{2} = \frac{1}{3} = 33,\overline{3}\%$$

Somit ergibt sich insgesamt folgende Ähnlichkeit für P_1 und P_2 bezüglich der Daten- und Datenflussperspektive:

$$SimData(P_1, P_2) = \frac{SimData_{P_1,P_2}(A) + SimData_{P_1,P_2}(B)}{2} = \frac{0,5 + \frac{1}{3}}{2} = \frac{\frac{5}{6}}{2} = \frac{5}{12} = 41,\overline{6}\%$$

Eine Ähnlichkeit von $41,\overline{6}\%$ lässt vermuten, dass die beiden Prozess P_1 und P_2 recht unähnlich bezüglich der Daten- und Datenflussperspektive sind. Dies lässt sich alleine schon aufgrund von Aktivität B begründen: Hier wären größere Anpassungen nötig, um die beiden Prozesse aneinander anzugleichen, da es kein einziges gemeinsames Output-Datenobjekt gibt. Dies hat

zur Folge, dass es insgesamt nur eine Ähnlichkeit von $33,\overline{3}\%$ für Aktivität B gibt. Für eine gründliche Interpretation der hier vorgestellten Verfahren und Berechnungen wird auf Kapitel 3.3.4 verwiesen.

3.3.4 Erläuterungen zu den entwickelten Ansätzen

Die in den Kapiteln 3.3.1-3.3.3 entwickelten Ansätze zum Vergleich der organisatorischen Perspektive, der operationalen Perspektive und der Daten- und Datenflussperspektive besitzen eine Gemeinsamkeit: Alle Ansätze liefern einen quantitativen Wert zwischen 0 und 1. Die jeweiligen Werte liefern dabei lediglich Ansatzpunkte zur Einschätzung der Gleichheit der beteiligten Modelle. So lässt ein großer Wert (z.B. $0,95$) auf relativ große Gleichheit schließen, wohingegen ein kleiner Wert (z.B. $0,1$) die Vermutung nahe legt, dass die Prozessmodelle sich stark unterscheiden.

Die Interpretation mittlerer Werte erfordert ein differenziertes Verständnis. In diesem Bereich spielt die fachkundige Einschätzung einer Expertin oder eines Experten eine entscheidende Rolle. Ein mittlerer numerischer Wert bedeutet nicht zwangsläufig eine mittlere Ähnlichkeit. Vielmehr können subtile Nuancen und Kontextabhängigkeiten dazu führen, dass ein mittlerer Wert sowohl auf relevante Gemeinsamkeiten als auch auf bedeutende Unterschiede hinweist. In solchen Fällen wird die Expertise benötigt, um die spezifischen operationalen, organisatorischen und datenbezogenen Kontexte berücksichtigen zu können.

Die Erkenntnisse dieses Kapitels haben direkte Auswirkungen auf die praktische Anwendung von Vergleichsansätzen. Unternehmen und Organisationen sollten nicht nur auf numerische Ähnlichkeitswerte vertrauen, sondern diese in den spezifischen Kontext ihrer Anwendungen einbetten. Insbesondere im Umgang mit mittleren Werten wird die enge Zusammenarbeit mit Fachleuten empfohlen. Dies ermöglicht eine kontextsensitive Interpretation und fördert ein ausgewogenes Verständnis der systemischen Ähnlichkeit bezüglich der relevanten Perspektiven.

3.4 Prozessmodellvergleich in verwandten Arbeiten

Die Feststellung von Ähnlichkeiten und gemeinsamen Merkmalen von Prozessmodellen ist von sehr großer Bedeutung sowohl in der Industrie als auch in der Forschung [60, 61]. Es ist einerseits unerlässlich, Duplikate [62] und verschiedene Modellvarianten [63] zu identifizieren, die auftreten können, wenn Prozessmodelle geändert oder entwickelt werden. Ein weiterer Schwerpunkt liegt auf der Korrektheit von Modellen und zielt darauf ab, die Überprüfung von Transformationen von Prozessmodellen zwischen verschiedenen Modellierungssprachen [64] sowie die Unterstützung der theoretischen Korrektheit von Prozessmodellen [65] sicherzustellen. Dabei werden in der Literatur im Allgemeinen vier verschiedene Dimensionen von Gleichheit und Ähnlichkeit für Prozessmodelle unterschieden [66]:

- Die *(natürliche) Sprachdimension* (engl. *natural language dimension*) beschreibt die Identifizierung gleicher Objekte, Aktivitäten etc.
- Zum Vergleich der *Graphdimension* (engl. *graph structure dimension*) werden Prozessmodelle auf gleiche Strukturen im korrespondierenden Graphen (z.B. Subgraphen) untersucht.
- Die *Verhaltensdimension* (engl. *behavioral dimension*) untersucht Prozessmodelle auf Basis von Sprachvergleichen.
- Die *(menschliche) Einschätzungsdimension* (engl. *human estimation dimension*) basiert auf der subjektiven Einschätzung von Prozessexpertinnen bzw. Prozessexperten.

Die Sprachdimension beschreibt im Prinzip den Vergleich der funktionalen Perspektive (Kapitel 3.1). Dieser geschieht meistens durch die Einschätzung von Domänenfachleuten oder durch Ansätze aus dem Bereich des Natural Language Processing [52, 53, 54]. Da der Vergleich der funktionalen Perspektive hauptsächlich dazu dient, Prozessmodelle grundsätzlich vergleichbar zu machen, wird darauf in dieser Arbeit ebenso wie in der Literatur nicht weiter eingegangen.

Die Graphdimension untersucht Prozessmodelle auf Ähnlichkeiten in der zugehörigen Graphstruktur. Dieser Ansatz bringt einerseits das Problem mit sich, dass nicht alle Prozessmodellierungssprachen eine grafische Repräsentation besitzen. Beispielsweise verfügt die deklarative Modellierungssprache DPIL [40, 41] über keine grafische Darstellungsform. Ein Vergleich bezüglich Graphstrukturen wäre hier demnach hinfällig. Außerdem kann die grafische Struktur irreführend sein: Zwei Prozessmodelle, die oberflächlich betrachtet ähnlich erscheinen, können tatsächlich sehr unterschiedlich sein [60]. Nichtsdestotrotz kann der Vergleich der Graphstrukturen Indikatoren liefern, um Ähnlichkeiten für Prozessmodelle zu quantifizieren [60]. In dieser Arbeit wird die Graphstruktur dennoch nicht weiter betrachtet, da qualitative Konzepte und Methoden zum Vergleich von Prozessmodellen völlig unabhängig von der verwendeten Modellierungssprache entwickelt werden sollten.

Im Bereich des Vergleichs der Verhaltensdimension werden die zu Prozessmodellen gehörenden Sprachen auf theoretischer Ebene untersucht. Dieser Bereich ist auch der in der Literatur am meisten verfolgte Ansatz zum Vergleich von Prozessmodellen [60]. So gibt es beispielsweise Ansätze zum Vergleich der Komplexitäten der Sprachen [67]. Damit ist es grundsätzlich möglich, zu bestimmen, ob ein Prozessmodell „komplexer“ ist als ein anderes. Dies kann beispielsweise bedeuten, dass es mehr Schleifen oder Verzweigungen besitzt. Dennoch ist für den Vergleich von Prozessmodellen nicht deren Komplexität der essentiellste Aspekt sondern vielmehr das konkrete Verhalten in der Form von schließlich ausgeführten Sequenzen [60].

Die letzte Dimension, die Einschätzungsdimension, beschäftigt sich mit der subjektiven Einschätzung von Prozessexpertinnen bzw. Prozessexperten und/oder Domänenexpertinnen bzw.

Domänenexperten. Natürlich kann auch ein geschulter Blick unter Umständen viele Erkenntnisse über verschiedene Prozessmodelle liefern und einschätzen, ob überhaupt und inwiefern sich die zugrunde liegenden Prozesse ähneln. Dennoch ist ein automatisiertes Verfahren erstrebenswerter, da zum einen nicht immer eine Prozessexpertin oder ein Prozessexperte verfügbar ist, zum anderen aber automatisierte Verfahren deutlich weniger fehleranfällig und im Regelfall auch effizienter sind.

Es bleibt also festzuhalten, dass die Verhaltensdimension die wichtigste und aussagekräftigste Dimension für den Prozessmodellvergleich ist. Daher fokussiert sich die vorliegende Dissertation auch auf diese Dimension.

Die aktuellen Herangehensweisen lassen sich in zwei Kategorien unterteilen: Ansätze zur imperativen Prozessmodellierungssprache und Ansätze zur deklarativen Prozessmodellierungssprache. Während es für imperative Prozessmodellierungssprachen eine Vielzahl von unterschiedlichen Ansätzen gibt [68, 69, 70], ist die Anzahl der Methoden für deklarative Prozessmodelle vergleichsweise gering. Die Studie in [17] zeigt auf, dass deklarative Prozessmodelle mit wenigen Constraints gut analysiert werden können, während die Behandlung von Modellen, die aus einer Vielzahl von Constraints bestehen, erhebliche Herausforderungen mit sich bringt. Diese mit der Anzahl der Constraints steigende Komplexität der Modelle ist zweifellos eine der Ursachen für diese Begrenzung. Gleichwohl gibt es vereinzelte Ansätze für deklarative Sprachen. In [15] wird beispielsweise ein automatenbasiertes Verfahren vorgestellt, um redundante Constraints und Widersprüche zwischen verschiedenen Constraints zu erkennen. Häufig werden alle Sequenzen bis zu einer bestimmten Länge simuliert, um ein besseres Verständnis der Prozessmodelle zu erlangen und mögliche Deadlocks zu identifizieren [25]. Dies ist auch die Standardvorgehensweise, um deklarative Prozessmodelle auf gegenseitige Teilmengenbeziehungen zu überprüfen [71]. Des Weiteren gibt es Ansätze, um hierarchische Beziehungen zwischen deklarativen Prozessmodellen zu identifizieren und zu überprüfen [43]. Bei allen Ansätzen, die Sequenzen bis zu einer gewissen Länge simulieren, bleibt jedoch offensichtlich das Restrisiko, dass diese Länge nicht ausreicht und bei längeren Sequenzen Widersprüche auftauchen könnten. Die vorliegende Arbeit liefert indes zwei Vergleichsmethoden für Prozessmodelle (Kapitel 3.2.4 und 3.2.6), die eine garantierte Aussage über Gleichheit bzw. Ungleichheit liefern.

Aufgrund des Mangels an Ansätzen für deklarative Prozessmodellierungssprachen gibt es so gut wie keine Methoden für den Vergleich von imperativen und deklarativen Prozessmodellen. In [72] werden das imperative und das deklarative Modellierungsparadigma auf einer sehr abstrakten Ebene allgemein verglichen. Ein solches Vorgehen erscheint jedoch wenig hilfreich, wenn es darum geht, die Ausdruckskraft konkreter Prozessmodelle zu untersuchen. In [73] definieren die Autoren die Gleichheit zwischen zwei Prozessmodellen (ungeachtet ihrer imperativen oder deklarativen Natur) basierend auf allen möglichen Ausführungssequenzen. Es bleibt jedoch unklar, wie das

Problem einer potenziell unendlichen Menge an Sequenzen gehandhabt werden soll, was ein offensichtliches Merkmal deklarativer Prozessmodelle darstellt und daher berücksichtigt werden muss.

Abschließend bleibt festzuhalten, dass es kaum Methoden zum Abgleich von multi-perspektivischen Prozessmodellen gibt. Alle zuvor genannten Vergleichsmethoden beziehen sich ausschließlich auf die funktionale Perspektive und die Kontrollflussperspektive. In der Literatur gibt es durchaus einige Ansätze für den Abgleich weiterer Perspektiven, z.B. [56, 57], die allerdings nur den direkten Vergleich von Elementen erlauben, ohne dabei das komplette Prozessmodell mit einzubeziehen. So werden unter anderem die Bausteine der Prozessmodelle paarweise miteinander verglichen. Dabei werden aber sämtliche potentielle Abhängigkeiten zwischen diesen ignoriert.

In dieser Dissertation werden innovative Ansätze entwickelt, um bestehende Forschungslücken im Vergleich von imperativen und deklarativen Prozessmodellen zu schließen. Die vorgestellte Methode trägt dazu bei, die Herausforderungen der begrenzten Vergleichsmethoden für deklarative Modelle zu überwinden. Durch die Berücksichtigung aller Perspektiven und die Fokussierung auf die Verhaltensdimension bietet diese Arbeit eine umfassende Grundlage für den Vergleich von Prozessmodellen. Die entwickelten Ansätze ermöglichen nicht nur eine präzise Analyse von Sequenzen und Constraints in deklarativen Modellen, sondern adressieren auch die bisher größtenteils vernachlässigte multi-perspektivische Betrachtung. Somit wird ein wichtiger Beitrag zur Entwicklung allgemein anwendbarer Methoden für den Vergleich von Prozessmodellen geleistet, um den Anforderungen in Industrie und Forschung gerecht zu werden.

Kapitel 4

Verhaltensänderungen in deklarativen Prozessmodellen

Aufgrund sich ständig ändernder Anforderungen an Geschäftsprozesse müssen Prozessmodelle regelmäßig überarbeitet werden, um sicherzustellen, dass sie noch aktuell sind und den zu Grunde liegenden Prozess tatsächlich widerspiegeln. Doch gerade im Bereich der deklarativen Prozessmodellierung sind Änderungen an Modellen oftmals schwierig umzusetzen, oder aber es treten ungewollte Änderungen auf. In diesem Kapitel wird eine Methode vorgestellt, mit der Verhaltensänderungen in Prozessmodellen, die durch vorgenommene Änderungen am Modell entstanden sind, effizient berechnet werden können. Diese Methode bezieht sich auf den Schritt *Modelle analysieren* des Gesamtkonzepts in Abbildung 1.5 und hilft beim Überarbeiten von Modellen bzw. Verifizieren von durchgeführten Änderungen. Große Teile dieses Abschnitts sind bereits in Publikation [23] veröffentlicht. Als einführendes Beispiel dient das Declare-Prozessmodell

$$M = (\{A, B, C, D, E\}, \{\text{response}(A, B), \text{response}(B, C), \text{response}(D, E)\})[23].$$

Dieses Modell steht repräsentativ für ein beliebiges Declare-Modell, auf welches das in diesem Abschnitt entwickelte Konzept gleichermaßen anwendbar ist. M wird nun verändert, indem das dritte Constraint ($\text{response}(D, E)$) gelöscht und ein neues Constraint ($\text{response}(A, E)$) hinzugefügt wird [23]. Das neu entstandene Prozessmodell wird im Folgenden mit M' bezeichnet. Abbildung 4.1 zeigt die beiden Modelle M und M' sowie einige von ihnen akzeptierte und nicht-akzeptierte Beispielsequenzen. Man erkennt sofort, dass durch die getätigten Änderungen auch Verhaltensänderungen auftreten. So war beispielsweise die Sequenz $\langle ABC \rangle$ im ursprünglichen Modell M erlaubt. Durch das neu hinzugefügte Constraint $\text{response}(A, E)$ wird jedoch in Modell M' zusätzlich, falls Aktivität A ausgeführt wird, das Ausführen von Aktivität E gefordert. Somit ist aufgrund des Fehlens von Aktivität E die Sequenz $\langle ABC \rangle$ in Modell M' nicht mehr

Modell M	**Geändertes Modell M'**	**Erlaubtes Verhalten (Beispiele)**
response(A, B)	response(A, B)	Verhalten von M:
response(B, C)	response(B, C)	✓ABC ✓BC ✗$AEBDC$ ...
response(D, E)	~~response(D, E)~~	Verhalten von M':
	response(A,E)$^{(NEU)}$	✗ABC ✓BC ✓$AEBDC$ ...

Abbildung 4.1: Prozessmodell M, geändertes Modell M' und Beispiele akzeptierter (✓) sowie nicht-akzeptierter (✗) Sequenzen [23]

möglich. Andererseits ist die Sequenz $\langle AEBDC \rangle$ in Modell M auf Grund des response(D, E) Constraints nicht erlaubt. Hier müsste nach der Ausführung von Aktivität D zu einem beliebigen Zeitpunkt noch die Ausführung von Aktivität E folgen. Durch das Löschen dieses Constraints ist die Sequenz $\langle AEBDC \rangle$ allerdings wieder erlaubt. Auch das Hinzufügen des response(A, E) Constraints besitzt keinen Einfluss auf die Akzeptanz der Sequenz $\langle AEBDC \rangle$. Folglich wird $\langle AEBDC \rangle$ von Prozessmodell M' akzeptiert.

Für das langfristige Management von Modellen ist das Verständnis obiger Veränderungen von größter Bedeutung, da Modelliererinnen und Modellierer sich darüber bewusst sein müssen, ob die Änderung des Prozessmodells zu einem nicht-konformen Verhalten geführt hat oder umgekehrt, ob wichtiges Verhalten nicht mehr unterstützt wird [74]. Selbst bei derartig kleinen Beispielen kann es jedoch durchaus unpraktikabel sein, alle möglichen Änderungen manuell zu berücksichtigen. Hierfür sind Ansätze erforderlich, die Modelliererinnen und Modellierer beim Verständnis von Verhaltensänderungen unterstützen. In diesem Kapitel wird ein Ansatz zur Berechnung von Verhaltensänderungen entwickelt, der konkrete Mengen an Sequenzen liefert, in denen sich das ursprüngliche Modell M von der abgeänderten Version M' unterscheidet.

Die Berechnung der Verhaltensänderungen erfolgt in vier Schritten, die in den folgenden Unterkapiteln genauer erläutert werden. Ein Überblick über das Gesamtkonzept zur Berechnung von Verhaltensänderungen ist in Abbildung 4.2 dargestellt. Im ersten Schritt erfolgt die Transformation deklarativer Prozessmodelle in endliche deterministische Automaten (Kapitel 4.1). Im zweiten Schritt werden mithilfe dieser deterministischen Automaten die beiden Differenzautomaten für die jeweiligen Differenzsprachen berechnet (Kapitel 4.2). Diese Differenzautomaten dienen im dritten Schritt zur Ermittlung sämtlicher auftretenden Verhaltensänderungen bis zu einer bestimmten Sequenzlänge (Kapitel 4.3). Die berechneten Verhaltensänderungen werden abschließend der Benutzerin oder dem Benutzer präsentiert (Kapitel 4.4). Im letzten Unterkapitel 4.5 werden abschließend noch verwandte Arbeiten zur Berechnung von Verhaltensänderungen diskutiert.

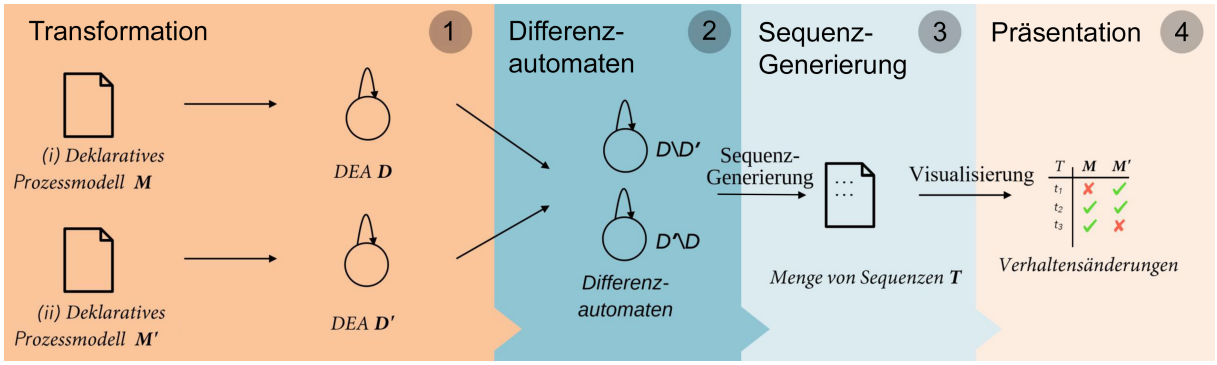


Abbildung 4.2: Konzept zur Berechnung von Verhaltensänderungen [23]

4.1 Transformation

Zunächst werden die beiden Prozessmodelle M und M' unter Anwendung der Tansformationsregeln aus Kapitel 3.2.2 in Prozessautomaten D und D' transformiert (Schritt 1 in Abbildung 4.2). Abbildung 4.3 zeigt die beiden Prozessautomaten zu den beiden Modellen M und M' aus Abbildung 4.1. Ein erster Blick auf diese Automaten zeigt bereits, dass sich auf jeden Fall am Verhalten der Modelle etwas geändert hat: Der Prozessautomat zum neuen Modell M' akzeptiert beispielsweise die alleinige Ausführung von Aktivität D, während $\langle D \rangle$ keine valide Sequenz des Prozessautomaten zum Modell M ist.

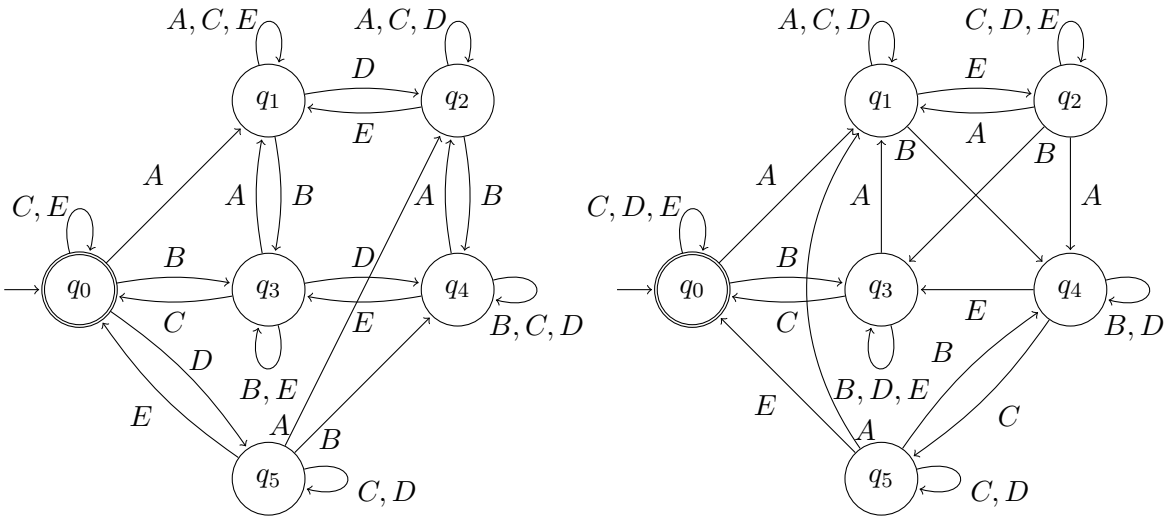


Abbildung 4.3: Prozessautomaten D (links) und D' (rechts) für Prozessmodelle M und M' [23]

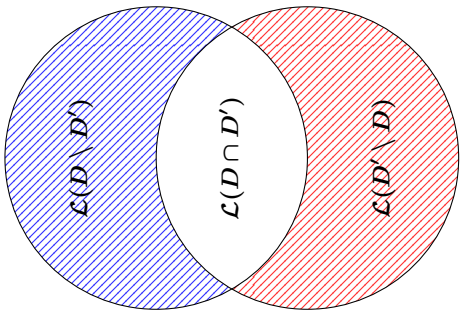


Abbildung 4.4: Venn-Diagramm für Verhaltensänderungen [23]

4.2 Berechnung der Differenzautomaten

Jetzt werden die beiden Differenzautomaten $\text{Diff}_{D,D'} := D \setminus D'$ und $\text{Diff}_{D',D} := D' \setminus D$ (Kapitel 3.2.9) berechnet (Schritt 2 in Abbildung 4.2). Deren Bedeutungen im Bezug auf Verhaltensänderungen werden im Folgenden interpretiert [23]:

$\text{Diff}_{D,D'}$: Dieser Differenzautomat akzeptiert alle Sequenzen, die von D akzeptiert werden, jedoch nicht von D', d.h. $\mathcal{L}(\text{Diff}_{D',D}) = \mathcal{L}(D) \setminus \mathcal{L}(D')$ (siehe blau schraffierte Menge in Abbildung 4.4). Diese entsprechen also genau den Sequenzen, die durch die Änderung am Modell verloren gegangen sind. Somit beschreibt der Differenzautomat $\text{Diff}_{D,D'}$ das sogenannte *verloren gegangene Verhalten* $T_{lost} := \mathcal{L}(\text{Diff}_{D,D'})$.

$\text{Diff}_{D',D}$: Dieser Differenzautomat akzeptiert alle Sequenzen, die von D' akzeptiert werden, jedoch nicht von D, d.h. $\mathcal{L}(\text{Diff}_{D,D'}) = \mathcal{L}(D') \setminus \mathcal{L}(D)$ (siehe rot schraffierte Menge in Abbildung 4.4). Diese entsprechen also genau den Sequenzen, die durch die Änderung am Modell neu hinzugekommen sind. Somit beschreibt der Differenzautomat $\text{Diff}_{D',D}$ das sogenannte *hinzugefügte Verhalten* $T_{added} := \mathcal{L}(\text{Diff}_{D',D})$.

Abbildung 4.5 zeigt den Differenzautomaten $\text{Diff}_{D',D}$ für das Beispiel aus Abbildung 4.1. Dieser beschreibt die durch Löschen des response(D, E) und Hinzufügen des response(A, E) Constraints weggefallenen Sequenzen. So gilt beispielsweise $\langle ABC \rangle \in \mathcal{L}(\text{Diff}_{D',D})$, was mit der Beobachtung in der dritten Spalte in Abbildung 4.1 einhergeht. Das ursprüngliche Modell M akzeptiert diese Sequenz, wohingegen sie vom neuen, geänderten Modell nicht akzeptiert wird.

Der Differenzautomat $\text{Diff}_{D,D'}$, der alle neu hinzugekommenen Sequenzen für das Beispiel aus Abbildung 4.1 beschreibt, ist in Abbildung 4.6 dargestellt. Man sieht, dass zum Beispiel $\langle AEBDC \rangle \in \mathcal{L}(\text{Diff}_{D',D})$ gilt. Diese Beobachtung bestätigt auch die dritte Spalte in Abbildung 4.1. War $\langle AEBDC \rangle$ im ursprünglichen Prozessmodell M keine valide Sequenz, wird sie vom aktualisierten Modell M' nun akzeptiert.

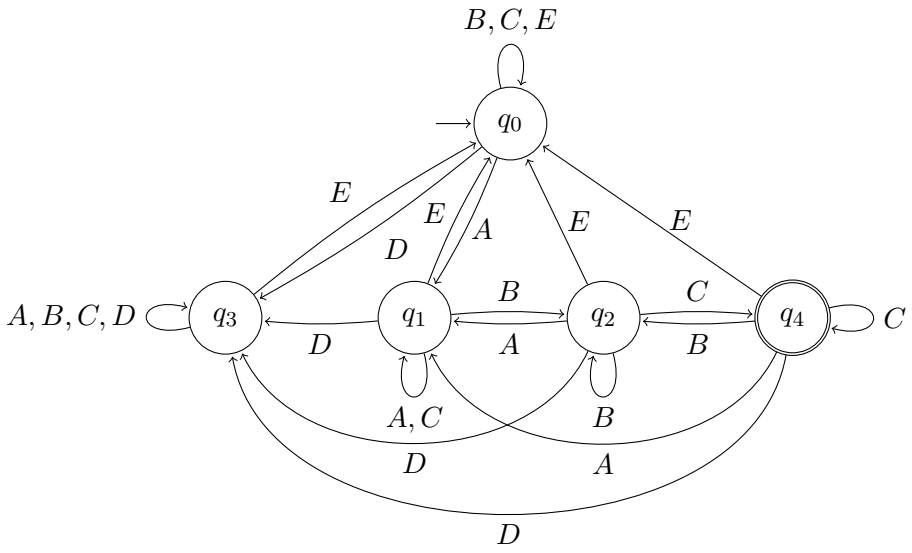


Abbildung 4.5: Differenzautomat $\text{Diff}_{D,D'}$ [23]

4.3 Sequenz-Generierung

Da die Sprache eines deterministischen endlichen Automaten in der Regel unendlich ist [50], sind auch die Verhaltensänderungen bei Prozessmodellen unter Umständen unendlich. Daher werden nun alle Änderungen bis zu einer gewissen Sequenzlänge l ermittelt (Schritt 3 in Abbildung 4.2). Diese beliebige Obergrenze l kann dabei von der Benutzerin oder vom Benutzer frei gewählt und je nach Anwendungsfall beliebig angepasst werden. Dafür werden alle von den beiden Differenzautomaten $\text{Diff}_{D,D'}$ und $\text{Diff}_{D',D}$ akzeptierten Wörter bis zur Länge l berechnet. Die Berechnung der Wörter folgt analog zur Berechnung der Vervollständigungen der Sequenzen in Kapitel 6.2 durch Anwenden einer Breitensuche [75] auf den jeweiligen Differenzautomaten (beginnend im Startzustand des jeweiligen Differenzautomatens). Somit erhält man zwei endliche Mengen, die die Verhaltensänderung bis zur Länge l beschreiben. Eine Menge beschreibt das verloren gegangene Verhalten T^l_{lost}, die andere das neu hinzugefügt Verhalten T^l_{added}, kurz [23]:

$$T^l_{lost} := \{t \in \mathcal{L}(\text{Diff}_{D,D'}) \mid |t| \leq l\}$$
$$T^l_{added} := \{t \in \mathcal{L}(\text{Diff}_{D',D}) \mid |t| \leq l\}$$

Angewandt auf das Beispiel aus Abbildung 4.1 erhält man die Mengen T^5_{lost} und T^5_{added} in Tabelle 4.1. Diese repräsentieren die durch die Veränderung verloren gegangenen und neu hinzugefügten Sequenzen. Beim hinzugefügten Verhalten T^5_{added} gibt es 64 Sequenzen der Länge 4 und 262 Sequenzen der Länge 5. Daher werden hier nur Sequenzen bis zur Länge $l = 3$ dargestellt.

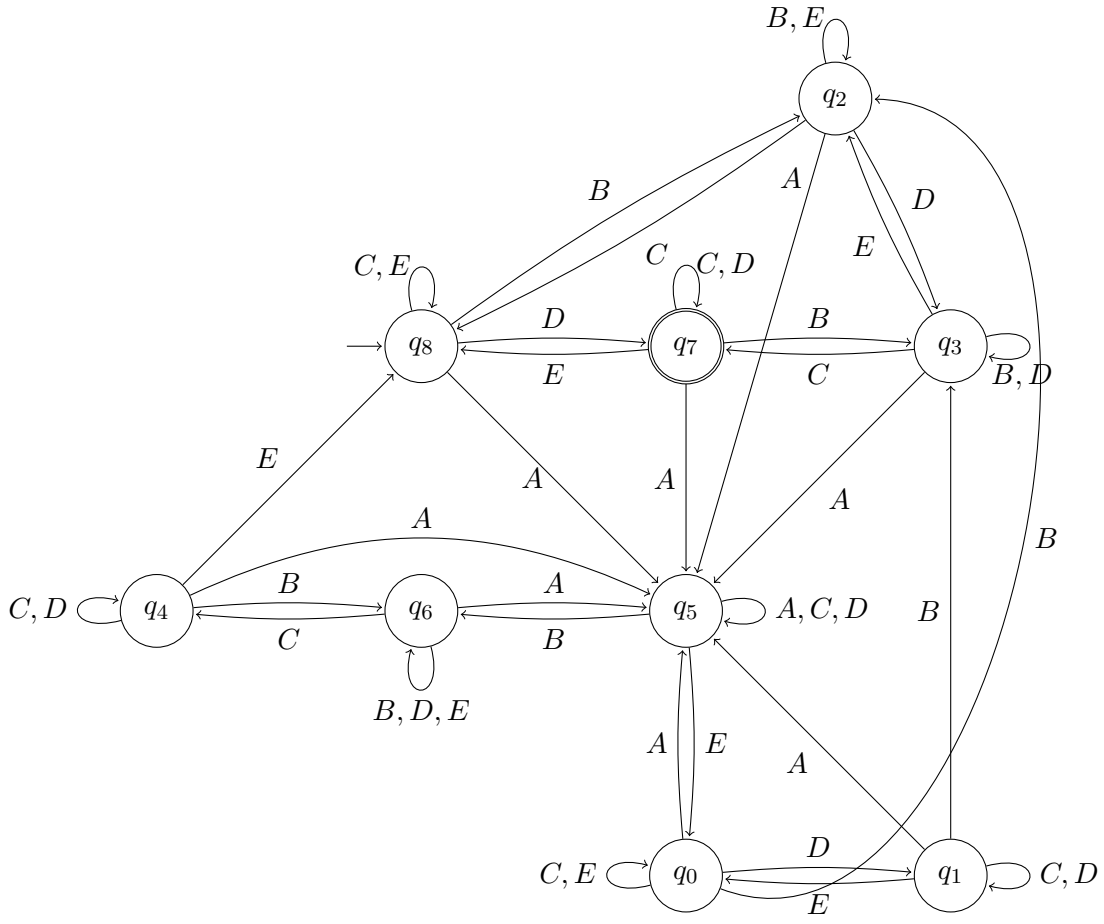


Abbildung 4.6: Differenzautomat $\text{Diff}_{D',D}$

4.4 Präsentation der Verhaltensänderungen

Im Folgenden werden der Nutzerin oder dem Nutzer die berechneten Verhaltensänderungen in einer geeigneten Weise präsentiert (Schritt 4 in Abbildung 4.2). Eine Möglichkeit der Darstellung der Verhaltensänderungen wäre, alle Sequenzen, die entweder gelöscht wurden oder dazugekommen sind, aufzulisten und dabei jeweils anzugeben, in welchem Modell sie erlaubt sind. Dies ist für obiges Beispiel in Tabelle 4.2 dargestellt.

Da die Anzahl der verloren gegangenen oder neu hinzugekommenen Sequenzen oft in die Tausende oder gar Hunderttausende steigt (Kapitel 8.2), ist es in der Regel nicht sinnvoll,

T^5_{lost}	T^5_{added}
$\langle ABC \rangle$,	$\langle D \rangle$,
$\langle CABC \rangle, \langle ABBC \rangle, \langle ACBC \rangle, \langle EABC \rangle, \langle BABC \rangle, \langle AABC \rangle, \langle ABCC \rangle,$	$\langle DD \rangle, \langle CD \rangle, \langle DC \rangle, \langle ED \rangle$
$\langle EACBC \rangle, \langle BABCC \rangle, \langle BBABC \rangle, \langle CABCC \rangle, \langle CBABC \rangle, \langle EAABC \rangle,$	$\langle BDC \rangle, \langle BCD \rangle, \langle CDD \rangle, \langle DDC \rangle, \langle DED \rangle, \langle CDC \rangle, \langle CED \rangle,$
$\langle ECABC \rangle, \langle AAABC \rangle, \langle ABCCC \rangle, \langle ACCBC \rangle, \langle ACBBC \rangle, \langle ABBCC \rangle,$	$\langle DBC \rangle, \langle DCD \rangle, \langle EDD \rangle, \langle CCD \rangle, \langle DCC \rangle, \langle DDD \rangle, \langle EDC \rangle,$
$\langle ACABC \rangle, \langle AACBC \rangle, \langle BEABC \rangle, \langle BCABC \rangle, \langle BAABC \rangle, \langle BABBC \rangle,$	$\langle EED \rangle, \langle ECD \rangle$
$\langle BACBC \rangle, \langle CACBC \rangle, \langle AEABC \rangle, \langle CAABC \rangle, \langle CABBC \rangle, \langle EABCC \rangle,$	+64 Sequenzen der Länge 4
$\langle CCABC \rangle, \langle EBABC \rangle, \langle AABCC \rangle, \langle ABBBC \rangle, \langle CEABC \rangle, \langle ABCBC \rangle,$	+262 Sequenzen der Länge 5
$\langle ABABC \rangle, \langle ACBCC \rangle, \langle DEABC \rangle$	

Tabelle 4.1: Beispiele für aus Änderungen am Modell M resultierende Verhaltensänderungen

Sequenz	M	M'
$\langle ABC \rangle$	✓	✗
$\langle AABC \rangle$	✓	✗
...	✓	✗
$\langle D \rangle$	✗	✓
$\langle DBC \rangle$	✗	✓
...	✗	✓

Tabelle 4.2: Darstellung der Verhaltensänderungen zwischen M und M' für eine Sequenzlänge von 4 (8 Sequenzen gehen verloren, 85 Sequenzen werden hinzugefügt) [23]

jede einzelne Sequenz zu untersuchen, da es für Menschen nicht möglich ist, eine derart hohe Anzahl an Sequenzen manuell zu verarbeiten. Daher wird in diesem Fall empfohlen, sich nur auf eine Teilmenge der Sequenzen zu beschränken. Dafür gibt es unzählig viele Techniken aus dem Bereich des Process Mining. Hier wird als Beispiel das Konzept von *repräsentativen Sequenzen* (engl. *representative traces*) gewählt. Dieses wurde in [76] eingeführt. Unter Verwendung verschiedener Clustering-Methoden ermöglicht dieser Ansatz das Übergeben einer (endlichen) Menge von Sequenzen und gibt eine (viel) kleinere Menge von repräsentativen Sequenzen aus dieser Menge zurück [76]. Dieses Vorgehen ermöglicht es, der Fachkraft eine viel kleinere Liste von repräsentativen Sequenzen vorzulegen, d.h. eine Menge von Sequenzen, die für die Verhaltensänderungen repräsentativ sind [23].

Ein Beispiel für die Präsentation der Verhaltensänderungen ist in Abbildung 4.7 illustriert, die die repräsentativen Sequenzen für das Beispiel aus Abbildung 4.1 mit einer benutzerdefinierten Sequenzlänge von 8 ($>$ 22k Sequenzen) zeigt. Der linke Block repräsentiert dabei das durch die Änderung verloren gegangene Verhalten und der rechte Block das neu hinzugefügte Verhalten. Jede Zeile in einem Block entspricht einer Sequenz, deren Aktivitäten durch bunte Quadrate dargestellt sind. Beispielsweise repräsentiert die erste Zeile des linken Blocks die Sequenz $\langle CCEABCC \rangle$. An dieser Illustration lässt sich unter anderem erkennen, dass viele Sequenzen, die mit Aktivität C (symbolisiert durch das schwarze Quadrat) enden, verloren gegangen sind. Die Nutzerin oder

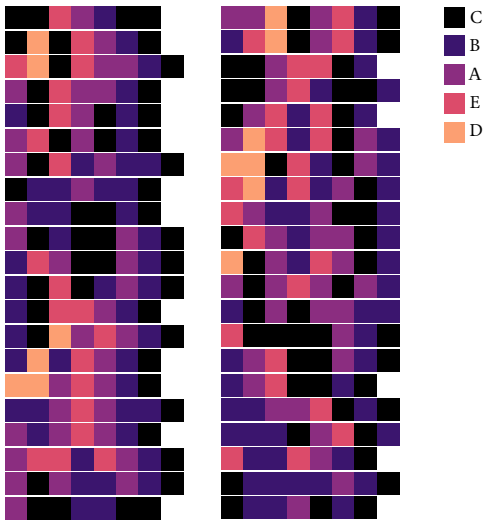


Abbildung 4.7: Repräsentative Sequenzen für verloren gegangenes (links) und neu hinzugefügtes (rechts) Verhalten zwischen M und M' bis zur Länge $l = 8$ (ungefähr 22k Sequenzen insgesamt) [23]

der Nutzer kann daher überprüfen, ob hier ein wichtiges Verhalten vorliegt, das beibehalten werden sollte. In diesem Fall wäre eine weitere Modellierung erforderlich. Ebenso gibt es viele neu hinzugefügte Sequenzen, die es ermöglichen, mit Aktivität B zu enden. Auch diese Information kann die Benutzerin oder der Benutzer verwenden, um zu prüfen, ob dies konform ist oder ob Aktivität C in der Sequenz auftreten sollte.

4.5 Berechnung von Verhaltensänderungen in verwandten Arbeiten

Im Wesentlichen ist das in diesem Kapitel entwickelte Verfahren zur Berechnung von Verhaltensänderungen eine Kombination aus mehreren in Abschnitt 3 eingeführten Methoden zum Prozessmodellvergleich. Hier wird das ursprüngliche Prozessmodell mit seiner abgeänderten Version verglichen. Somit sind alle in Kapitel 3.4 vorgestellten Ansätze aus der Literatur auch hier relevant.

Allerdings hebt sich der Fokus dieses Kapitels von allen anderen Ansätzen hinsichtlich der Anwendung ab, da hier versucht wird, die Unterschiede zwischen verschiedenen Modellen konkret zu berechnen. Bisher wurde diesem Thema, das sich direkt aus der langfristigen Verwaltung von deklarativen Prozessmodellen ableitet, nur wenig Beachtung geschenkt. Alle existierenden Arbeiten ermöglichen keine exakte Berechnung der Änderungen als Menge an Sequenzen. Dies wäre jedoch für Expertinnen und Experten von entscheidender Bedeutung, um ungewolltes

Verhalten entdecken und somit Fehler in der Modellierung beheben zu können.

Es gibt indes bereits einige Ansätze, die sich peripher mit dieser Thematik beschäftigen. So untersuchen unter anderem die Autoren in [77] die Auswirkungen manueller Änderungen an deklarativen Prozessmodellen hinsichtlich Verständlichkeit und Qualität der Modelle. Des Weiteren wird die Idee von Verhaltensänderungen zwischen zwei Modellen in [71] diskutiert. Eine Einschränkung dieser Arbeit besteht jedoch darin, dass eine Menge an Sequenzen, die verglichen werden soll, bereits zuvor bereitgestellt werden muss. Somit muss in der Praxis ein Unternehmen einen konkreten Satz an Sequenzen zur Verfügung stellen oder aber es müssen alle möglichen Sequenzen bis zu einer bestimmten Länge berücksichtigt werden. Letzteres ist offensichtlich aufgrund der exponentiellen Skalierung nicht praktikabel.

Zusammengefasst stellen die Identifizierung und Quantifizierung von Verhaltensänderungen zwischen verschiedenen Versionen deklarativer Prozessmodelle eine wesentliche Forschungslücke dar, die bislang nur begrenzt Beachtung gefunden hat. Die Notwendigkeit, nicht nur Änderungen zu erkennen, sondern sie auch als präzise Menge von Sequenzen zu berechnen, ist entscheidend für die langfristige Verwaltung solcher Modelle. Diese Forschungslücke wird jedoch durch den in diesem Kapitel entwickelten Ansatz erfolgreich geschlossen. Die vorgestellte Methode kombiniert verschiedene in Kapitel 3 eingeführte Methoden zum Prozessmodellvergleich und ermöglicht so eine präzise und anwendbare Berechnung von Verhaltensänderungen. Damit trägt diese Arbeit maßgeblich dazu bei, die bisherige Lücke in der Forschung hinsichtlich der exakten Berechnung von Änderungen zwischen deklarativen Prozessmodellen zu überwinden.

Kapitel 5

Ähnlichkeit von Prozessmodellen

Natürlich stellt sich bei ungleichen Prozessmodellen bzw. verschiedenen Modellvarianten unter anderem auch die Frage, wie groß der Unterschied zwischen den Modellen ist bzw. wie ähnlich sie sich sind. Um die Ähnlichkeit von Prozessmodellen zu bestimmen, hat man grundsätzlich zwei Möglichkeiten. Einerseits können die Strukturen der jeweiligen Prozessmodelle z.B. durch Graphentheorie untersucht werden. Andererseits besteht die Möglichkeit, Prozessmodelle im Hinblick auf deren Verhalten, d.h. ihre akzeptierten Wörter bzw. Sequenzen zu untersuchen. Aufgrund der in Kapitel 3.4 beschriebenen Gründe wird in dieser Arbeit der Fokus auf den Vergleich der Verhaltensweisen der Modelle, d.h. deren Sprachen, gelegt.

Da im Prozessmanagement lediglich Sequenzen von endlicher Länge in Frage kommen, ist es legitim, sich bei den korrespondierenden Wörtern der Prozessautomaten auf endliche Längen zu beschränken. Daher werden alle im Folgenden eingeführten Ähnlichkeitsmaße in Abhängigkeit von der Sequenzlänge definiert. Während die ersten vier definierten Ähnlichkeitsmaße (Kapitel 5.1-5.4) sich prinzipiell mit der Anzahl der Sequenzen, die von den beiden Modellen akzeptiert werden, beschäftigen, widmet sich Kapitel 5.5 einem Ähnlichkeitsmaß, welches das Hauptaugenmerk auf die Ähnlichkeit der verschiedenen Sequenzen legt. Abschließend werden in Kapitel 5.6 Ähnlichkeitsmaße in verwandten Arbeiten diskutiert. Alle Ähnlichkeitsmaße wurden vom Autor in Publikation [22] veröffentlicht.

In diesem Kapitel wird durchgehend der Begriff der *Menge der n-Sequenzen* eines Prozessmodells P verwendet. Dieser bezeichnet die Menge aller Sequenzen der Länge n, die von P akzeptiert werden. Dieser Begriff wird im Folgenden noch formal definiert:

Definition 22. *Sei P ein Prozessmodell. Für $n \in \mathbb{N}$ heißt*

$$P(n) := \{\sigma \in \mathcal{L}(P) \mid |\sigma| = n\} \subseteq \mathcal{L}(P)$$

die ***Menge der n-Sequenzen von P.***

Zum Abschluss sei darauf hingewiesen, dass sämtliche in diesem Kapitel entwickelten Ähnlichkeitsmaße lediglich als Indizien für die Ähnlichkeit von Prozessmodellen dienen. Dabei lässt sich keine präzise Grenze festlegen, die bestimmen könnte, ob gewisse Werte eine signifikante Ähnlichkeit aufweisen. In sämtlichen Anwendungsfällen bedarf es daher stets der Auslegung aller Ähnlichkeitswerte durch Fachleute.

5.1 n-Dichte

Als erstes Ähnlichkeitsmaß für Prozessmodelle wird die sogenannte *n-Dichte* [22] formuliert.

Definition 23. *Sei P ein Prozessmodell über einer endlichen Menge von Aktivitäten $\mathcal{A}$. Für $n \in \mathbb{N}$ heißt*

$$\lambda_n(P) := \frac{|P(n)|}{|\mathcal{A}|^n} \in [0,1]$$

*die n-**Dichte** von P.*

Da die Menge der n-Sequenzen $P(n)$ eine Teilmenge aller möglichen Sequenzen der Länge n bildet, nimmt die n-Dichte einen Wert zwischen 0 und 1 an. Sie beschreibt demnach den Prozentsatz aller Sequenzen einer gewissen Länge n, die von einem Prozessmodell P akzeptiert werden.

Im Folgenden wird Beispielprozess P_4 (Abbildung 3.17, Kapitel 3.2.5) näher betrachtet. P_4 besteht aus den drei Aktivitäten A, B und C und vier Constraints: succession(A, B), chainResponse(A, B), chainResponse(B, C) und respondedExistence(A, B). Tabelle 5.1 zeigt alle Sequenzen bis zur Länge $n = 8$, die von P_4 akzeptiert werden. Aus diesen Mengen können nun die n-Dichten bestimmt werden. Dabei steht im Nenner immer die n-te Potenz von 3, da das Prozessmodell P_4 drei Aktivitäten umfasst. Im Zähler findet man jeweils die Kardinalität der Menge der rechten Spalte aus Tabelle 5.1.

So ergibt sich beispielsweise $\lambda_0(P_4) = \frac{1}{3^0} = 1$ oder $\lambda_4(P_4) = \frac{3}{3^4} = \frac{3}{81} (= \frac{1}{27})$. Alle Werte der n-Dichte bis $n = 8$ sind in Tabelle 5.2 aufgelistet.

Man sieht, dass die n-Dichte für Prozessmodell P_4 streng monoton fallend ist, d.h. $\lambda_n(P_4) > \lambda_{n+1}(P_4)$ für alle $n \leq 7$. Es lässt sich vermuten, dass dies für alle $n \in \mathbb{N}$ gilt, da insbesondere durch die beiden chainResponse Constraints sehr viele mögliche Sequenzen das Prozessmodell nicht mehr erfüllen. So triggert z.B. die Ausführung der Aktivität A die unmittelbar folgende Ausführung der Teilsequenz $\langle BC \rangle$. Somit gibt es für die nächsten beiden Prozessschritte nur diese eine Möglichkeit anstatt $3^2 = 9$ Möglichkeiten. Analog triggert beispielsweise die Ausführung von Aktivität B die direkt anschließende Ausführung von Aktivität C. Deshalb wird die Anzahl der gültigen Sequenzen mit steigender Sequenzlänge verhältnismäßig immer geringer.

Sequenzlänge n	$\{\sigma \in \mathcal{L}(P_4) \mid \lvert\sigma\rvert = n\}$
0	$\{\varepsilon\}$
1	$\{\langle C\rangle\}$
2	$\{\langle CC\rangle\}$
3	$\{\langle ABC\rangle, \langle CCC\rangle\}$
4	$\{\langle ABCC\rangle, \langle CABC\rangle, \langle CCCC\rangle\}$
5	$\{\langle ABCBC\rangle, \langle CCABC\rangle, \langle ABCCC\rangle, \langle CABCC\rangle, \langle CCCCC\rangle\}$
6	$\{\langle ABCABC\rangle, \langle ABCBCC\rangle, \langle CCCABC\rangle, \langle ABCCBC\rangle, \langle ABCCCC\rangle, \langle CABCBC\rangle, \langle CABCCC\rangle,$ $\langle CCABCC\rangle, \langle CCCCCC\rangle\}$
7	$\{\langle ABCABCC\rangle, \langle ABCBCBC\rangle, \langle ABCBCCC\rangle, \langle ABCCABC\rangle, \langle ABCCBCC\rangle, \langle ABCCCBC\rangle,$ $\langle ABCCCCC\rangle, \langle ABCCCCC\rangle, \langle CABCABC\rangle, \langle CABCBCC\rangle, \langle CABCCBC\rangle, \langle CABCCCC\rangle$ $\langle CCABCBC\rangle, \langle CCABCCC\rangle, \langle CCCABCC\rangle, \langle CCCCABC\rangle, \langle CCCCCCC\rangle\}$
8	$\{\langle ABCABCBC\rangle, \langle ABCABCCC\rangle, \langle ABCBCABC\rangle, \langle ABCBCBCC\rangle, \langle ABCBCCBC\rangle,$ $\langle ABCBCCCC\rangle, \langle ABCCABCC\rangle, \langle ABCCBCBC\rangle, \langle ABCCBCCC\rangle, \langle ABCCCABC\rangle, \langle ABCCCBCC\rangle,$ $\langle ABCCCCBC\rangle, \langle ABCCCCCC\rangle, \langle CABCABCC\rangle, \langle CABCBCBC\rangle, \langle CABCBCCC\rangle, \langle CABCCABC\rangle,$ $\langle CABCCBCC\rangle, \langle CABCCCBC\rangle, CABCCCCC\rangle, \langle CCABCABC\rangle, \langle CCABCBCC\rangle, \langle CCABCCBC\rangle,$ $\langle CCABCCCC\rangle, \langle CCCABCBC\rangle, \langle CCCABCCC\rangle, \langle CCCCABCC\rangle, \langle CCCCCABC\rangle, \langle CCCCCCCC\rangle\}$

Tabelle 5.1: Von P_4 akzeptierte Sequenzen bis zur Länge $n = 8$

Sequenzlänge n	n-Dichte $\lambda_n(P_4)$
0	$\frac{1}{3^0} = 1$
1	$\frac{1}{3^1} = \frac{1}{3}$
2	$\frac{1}{3^2} = \frac{1}{9}$
3	$\frac{2}{3^3} = \frac{2}{27}$
4	$\frac{3}{3^4} = \frac{3}{81}$
5	$\frac{5}{3^5} = \frac{5}{243}$
6	$\frac{9}{3^6} = \frac{9}{729}$
7	$\frac{17}{3^7} = \frac{17}{2187}$
8	$\frac{29}{3^8} = \frac{29}{6561}$

Tabelle 5.2: n-Dichte $\lambda_n(P_4)$ für Prozessmodell P_4 bis zur Länge $n = 8$

Die n-Dichte kann verwendet werden, um einen ersten Eindruck von der Ähnlichkeit von verschiedenen Prozessmodellen zu gewinnen. Dies ist besonders dann der Fall, wenn sich die n-Dichten der beiden Prozessmodelle stark unterscheiden. Hierzu ein Beispiel: Besitzt ein Prozessmodell P_1 die n-Dichte $\lambda_n(P_1) = 0,1$ für ein $n \in \mathbb{N}$, während ein zweites Prozessmodell P_2 eine n-Dichte von $\lambda_n(P_2) = 0,7$ aufweist, unterscheiden sich die beiden Prozessmodelle ganz erheblich, da maximal $\frac{1}{7}$ der Sequenzen von P_2 auch von P_1 akzeptiert wird. Dieser maximale Wert von $\frac{1}{7}$ tritt dabei genau dann ein, wenn P_1 in P_2 enthalten ist. Dann unterscheiden sich P_1 und P_2 in mindestens 60% der Sequenzen, die von P_2 akzeptiert werden. Insgesamt kann also von „weit auseinander liegenden“ n-Dichten auf unähnliche Prozessmodelle geschlossen werden.

Auf der anderen Seite kann man bei ähnlichen bzw. sogar gleichen n-Dichten im Allgemeinen keine Rückschlüsse ziehen auf große Ähnlichkeit der Prozessmodelle: Selbst im Falle $\lambda_n(P_1) = 0,5 = \lambda_n(P_2)$ wäre es sogar möglich, dass P_1 und P_2 komplett disjunkt sind und somit keine einzige gemeinsame Sequenz besitzen.

Insgesamt können für die n-Dichte $\lambda_n(P_1, P_2)$ zweier Prozessmodelle P_1 und P_2 zwei verschiedene Fälle auftreten:

1. $\lambda_n(P_1) + \lambda_n(P_2) \leq 1$
2. $\lambda_n(P_1) + \lambda_n(P_2) > 1$

Zunächst wird der erste Fall betrachtet. Dabei gibt es drei mögliche Szenarien, die sich für n-Sequenzen ergeben können (obere Reihe in Abbildung 5.1):

- $P_1(n) \cap P_2(n) = \emptyset$, d.h. $P_1(n)$ und $P_2(n)$ sind disjunkt.
- $P_1(n) \subseteq P_2(n)$ oder $P_2(n) \subseteq P_1(n)$, d.h. die Menge der n-Sequenzen eines Prozessmodells ist in der Menge der n-Sequenzen des anderen Prozessmodells enthalten.
- $P_1(n) \cap P_2(n) \neq \emptyset$ und kein Prozessmodell ist im anderen enthalten. Dann gibt es sowohl n-Sequenzen, die nur von P_1 als auch solche, die nur von P_2 akzeptiert werden.

Im zweiten Fall ($\lambda_n(P_1) + \lambda_n(P_2) > 1$) gibt es zwei Szenarien (untere Reihe in Abbildung 5.1), die für n-Sequenzen auftreten können:

- Keine Menge der n-Sequenzen ist in der anderen Menge der n-Sequenzen enthalten. Dann gibt es sowohl n-Sequenzen, die nur von P_1 akzeptiert werden als auch solche, die nur von P_2 akzeptiert werden.
- $P_1(n) \subseteq P_2(n)$ oder $P_2(n) \subseteq P_1(n)$, d.h. die Menge der n-Sequenzen eines Prozessmodells ist in der Menge der n-Sequenzen des anderen Prozessmodells enthalten.

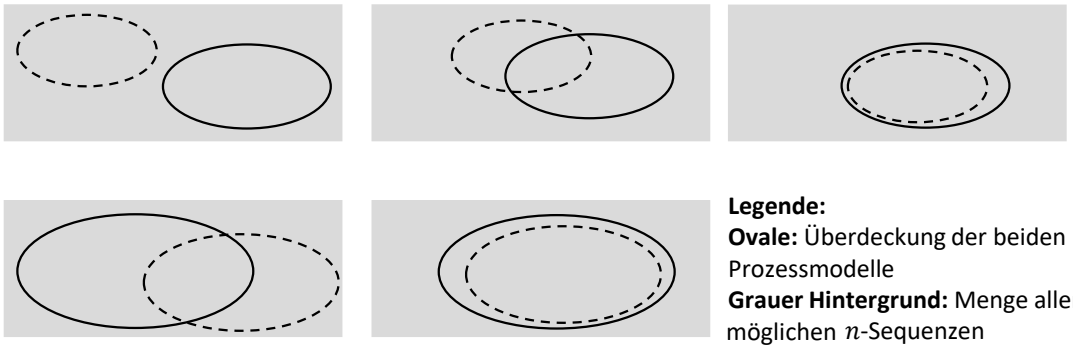


Abbildung 5.1: Darstellung aller Möglichkeiten der n-Dichte [22]

5.2 *min-max*-Dichte

In diesem Kapitel wird als weiteres Ähnlichkeitsmaß für Prozessmodelle die sogenannte *min-max-Dichte* eingeführt. Diese ist eine Verallgemeinerung der n-Dichte aus Kapitel 5.1. Zunächst wird diese formal definiert [22]:

Definition 24. *Sei P ein Prozessmodell über einer endlichen Menge von Aktivitäten $\mathcal{A}$. Für $min, max \in \mathbb{N}$ mit $min \leq max$ heißt*

$$\lambda_{min}^{max}(P) := \frac{\sum_{i=min}^{max} |P(i)|}{\sum_{i=min}^{max} |\mathcal{A}|^i} \in [0,1]$$

*die **min-max-Dichte** von P.*

Im Nenner steht die Anzahl aller möglichen Sequenzen der Längen zwischen min und max. Der Zähler gibt die Summe der Kardinalitäten der Mengen aller n-Sequenzen für $min \leq n \leq max$ an. Diese bilden, wie in Kapitel 5.1 bereits erwähnt, eine Teilmenge aller möglichen Sequenzen. Infolgedessen nimmt die *min-max*-Dichte ebenfalls einen Wert zwischen 0 und 1 an. Die *min-max*-Dichte ist eine Verallgemeinerung der n-Dichte auf Sequenzen der Längen in einem Intervall $[min, max]$. Für den Spezialfall $min = n = max$ erhält man folgende n-Dichte:

$$\lambda_n^n(P) = \frac{\sum_{i=n}^{n} |P(i)|}{\sum_{i=min}^{max} |\mathcal{A}|^i} = \frac{|P(n)|}{|\mathcal{A}|^n} = \lambda_n(P)$$

Oftmals ist man daran interessiert zu erfahren, wie sich Prozessmodelle bis zu einer gewissen Sequenzlänge n verhalten. Für $min = 0$ und $max = n$ betrachtet die 0-n-Dichte λ_0^n genau diesen Bereich.

Alle Überlegungen bezüglich der Ähnlichkeit von Prozessmodellen, die in Kapitel 5.1 zur n-Dichte gemacht wurden, können gleichermaßen auf die *min-max*-Dichte übertragen werden, da

max	$\lambda_0^{max}(P_4)$
0	$\frac{1}{1}$
1	$\frac{2}{4}$
2	$\frac{3}{13}$
3	$\frac{5}{20}$
4	$\frac{8}{101}$
5	$\frac{13}{344}$
6	$\frac{22}{1073}$
7	$\frac{39}{3260}$
8	$\frac{68}{9821}$

Tabelle 5.3: $\lambda_0^{max}(P_4)$ für Prozessmodell P_4 bis zur Länge $max = 8$

sich nur die „Grundmenge“ geändert hat: War es bei der n-Dichte die Menge der n-Sequenzen $P(n)$, ist es hier die Vereinigung der n-Sequenzen gewisser Längen:

$$\bigcup_{n \in \{min,\ldots,max\}} P(n)$$

Tabelle 5.3 zeigt alle Werte der 0-*max*-Dichte zu Prozessmodell P_4 (Abbildung 3.17 in Kapitel 3.2.5). Diese Werte untermauern die Beobachtungen zur n-Dichte aus Kapitel 5.1: Die 0-*max*-Dichte ist ebenso streng monoton fallend. Diese Eigenschaft ist naheliegend, da dies bereits für die n-Dichten gilt.

Je größer die Dichte innerhalb eines *min-max*-Bereichs in einem Prozessmodell P ist, desto mehr Sequenzen werden innerhalb von P ermöglicht. Diese erhöhte Dichte weist darauf hin, dass der zugrunde liegende Prozess eine höhere Flexibilität aufweist. In diesem Zusammenhang kann die *min-max*-Dichte als Maß für die Deklarativität dienen, wie in [11] vorgeschlagen. Wenn die *min-max*-Dichte relativ hoch ist, zeigt sich oft, dass eine deklarative Modellierungssprache für den vorliegenden Prozess besser geeignet ist als eine imperative Modellierungssprache und umgekehrt. Dies legt nahe, dass die *min-max*-Dichte als Indikator dafür fungieren kann, welcher Modellierungsansatz in einem bestimmten Kontext bevorzugt werden sollte.

5.3 n-Ähnlichkeit

Die beiden in den Kapiteln 5.1 (n-Dichte) und 5.2 (*min-max*-Dichte) vorgestellten Ähnlichkeitsmaße lieferten für jedes Prozessmodell P einen Wert, der angibt, welcher Prozentsatz aller möglichen

n-Sequenzen von P akzeptiert wird. Mit Hilfe der Werte für verschiedene Prozessmodelle konnten erste Aussagen über Ähnlichkeiten und Unterschiede von Prozessmodellen getroffen werden. Natürlich ist es wünschenswert, einen konkreten Wert für die Ähnlichkeit von Prozessmodellen zu haben. Im Folgenden wird ein Ähnlichkeitsmaß, das diese Anforderung erfüllt, eingeführt. Dieses Maß ist die sogenannte *n-Ähnlichkeit* [22]:

Definition 25. *Seien P_1 und P_2 Prozessmodelle über einer endlichen Menge von Aktivitäten. Für $n \in \mathbb{N}$ heißt*

$$\Lambda_n(P_1, P_2) := \frac{|P_1(n) \cap P_2(n)|}{|P_1(n)|} \in [0, 1]$$

*die **n-Ähnlichkeit** von P_1 und P_2.*

Vorab sei angemerkt, dass die n-Ähnlichkeit im Allgemeinen nicht symmetrisch ist, d.h. $\Lambda_n(P_1, P_2) \neq \Lambda_n(P_2, P_1)$. Zwar ist der Zähler symmetrisch, da der Durschnittsoperator kommutativ ist. Allerdings steht im Nenner die Mächtigkeit der n-Sequenzen des ersten Prozessmodells, was die Asymmetrie der n-Ähnlichkeit zur Folge hat.

Die n-Ähnlichkeit gibt an, welcher Prozentsatz der n-Sequenzen des Durchschnitts der beiden Prozessmodelle P_1 und P_2 von Prozessmodell P_1 akzeptiert werden. Somit werden mit Hilfe der n-Ähnlichkeit Vergleiche der Ausdrucksmächtigkeit der beiden Prozessmodelle möglich.

Der Hauptunterschied der n-Ähnlichkeit zu den Dichte-Maßen aus den Kapiteln 5.1 und 5.2 liegt darin, dass die Dichte-Maße das Verhältnis der n-Sequenzen, die von einem Prozessmodell akzeptiert werden, zu allen möglichen n-Sequenzen betrachten. Im Gegensatz dazu beschränkt sich die n-Ähnlichkeit auf die Menge der n-Sequenzen eines einzelnen Prozessmodells. Demzufolge liefert die n-Ähnlichkeit ein geeignetes Maß, um zu bestimmen, wie weit ein Prozessmodell vom Durchschnitt zweier Prozessmodelle abweicht.

Daraus lässt sich zudem folgern, dass eine höhere n-Ähnlichkeit eine größere Überlappung der beiden Prozessmodelle impliziert. Eine n-Ähnlichkeit von 0 bedeutet, dass P_1 und P_2 keine einzige gemeinsame n-Sequenz besitzen. Andererseits bedeutet eine n-Ähnlichkeit von 1, dass $P_1(n) \cap P_2(n) = P_1(n)$. Somit folgt, dass die Menge der n-Sequenzen von P_2 in der Menge der n-Sequenzen von P_1 enthalten ist, d.h. $P_2(n) \subseteq P_1(n)$.

Abschließend wird zur Veranschaulichung der n-Ähnlichkeit ein abstraktes Beispiel betrachtet. Sei dazu angenommen, dass Prozessmodell P_1 100 n-Sequenzen und Prozessmodell P_2 200 n-Sequenzen akzeptiert. Weiterhin bestehe die Menge der n-Sequenzen, die von P_1 und P_2 akzeptiert werden, aus 50 Elementen. Dann erhält man die beiden n-Ähnlichkeiten $\Lambda_n(P_1, P_2) = \frac{50}{100} = 0,5$ und $\Lambda_n(P_2, P_1) = \frac{50}{200} = 0,25$. Dies bedeutet, dass der Durchschnitt von $P_1(n)$ und $P_2(n)$ 50% von P_1 ausmacht, allerdings nur 25% von P_2. Demzufolge überdeckt Prozessmodell P_1 deutlich mehr Eigenschaften von Prozessmodell P_2 als umgekehrt.

5.4 *min-max*-Ähnlichkeit

Analog zur Erweiterung der n-Dichte (Kapitel 5.1) zur *min-max*-Dichte (Kapitel 5.2) wird nun das Ähnlichkeitsmaß der n-Ähnlichkeit (Kapitel 5.3) zur *min-max-Ähnlichkeit* erweitert. War man bei der n-Ähnlichkeit noch auf einzelne Sequenzlängen beschränkt, wird hier der Vergleich von Prozessmodellen in gewünschten Intervallen von Sequenzlängen möglich. Die *min-max*-Ähnlichkeit wird folgendermaßen formal definiert [22]:

Definition 26. *Seien P_1 und P_2 Prozessmodelle über einer endlichen Menge von Aktivitäten. Für $min, max \in \mathbb{N}$ mit $min \leq max$ heißt*

$$\Lambda_{min}^{max}(P_1, P_2) := \frac{\sum_{i=min}^{max} |P_1(i) \cap P_2(i)|}{\sum_{i=min}^{max} |P_1(i)|} \in [0, 1]$$

*die **min-max-Ähnlichkeit** von P_1 und P_2.*

Analog zur n-Ähnlichkeit ist die *min-max*-Ähnlichkeit nicht symmetrisch, da im Nenner nur Sequenzen eines Prozessmodells stehen. Alle Beobachtungen, die in Kapitel 5.3 zur n-Ähnlichkeit gemacht wurden, lassen sich gleichermaßen auf die *min-max*-Ähnlichkeit übertragen, da sich analog zur *min-max*-Dichte nur die „Grundmenge“ (d.h. der Nenner) geändert hat.

5.5 n-m-Damerau-Levenshtein-Ähnlichkeit

In den Abschnitten 5.1-5.4 wird die Ähnlichkeit von Prozessmodellen über die Anzahl der gemeinsam akzeptierten Sequenzen bestimmt. Dabei würden beispielsweise die beiden Sequenzen $\langle ABCDEF \rangle$ und $\langle ABCDEFG \rangle$ nicht berücksichtigt werden, da sie nicht gleich sind. Sie unterscheiden sich jedoch nur marginal, und oftmals ist dieser geringe Unterschied in der Form von nur einer Aktivität nicht wirklich ausschlaggebend. Somit wäre es sinnvoll, ein Ähnlichkeitsmaß zu besitzen, das auch „ähnliche“ Sequenzen berücksichtigt. Da die Sequenzen als Zeichenkette repräsentiert werden, ist folglich ein Maß zur Ähnlichkeit von Zeichenketten notwendig. In dieser Arbeit wird die *Damerau-Levenshtein-Distanz* [78] verwendet. Diese gewichtet vertauschte Zeichen nicht so stark wie andere Zeichenkettenvergleiche, was folgenden Vorteil liefert [79]: Die beiden Sequenzen $\langle AB \rangle$ und $\langle BA \rangle$ könnten äquivalent sein, da die beiden Aktivitäten A und B unter Umständen parallel durchgeführt werden und lediglich durch verschiedene Laufzeitbedingungen einmal $\langle AB \rangle$ und einmal $\langle BA \rangle$ dokumentiert wird. Dies sieht man den Sequenzen per se nicht mehr an. Deswegen möchte man diese beiden Sequenzen nicht so hart gewichten wie beispielsweise die Sequenzen $\langle AB \rangle$ und $\langle AA \rangle$, die sich in einem Symbol unterscheiden. Dazu wird die Damerau-Levenshtein-Distanz für zwei Sequenzen formal definiert [78]:

Definition 27. *Seien σ_1 und σ_2 Sequenzen, $i \in \mathbb{N}_{\leq|\sigma_1|}, j \in \mathbb{N}_{\leq|\sigma_2|}$ und d die folgendermaßen definierte rekursive Funktion:*

$$d_{\sigma_1,\sigma_2}(i,j) := min \begin{cases} 0 & \text{, falls } i = j = 0 \\ d_{\sigma_1,\sigma_2}(i-1,j)+1 & \text{, falls } i > 0 \\ d_{\sigma_1,\sigma_2}(i,j-1)+1 & \text{, falls } j > 0 \\ d_{\sigma_1,\sigma_2}(i-1,j-1)+\mathbf{1}_{\sigma_1(i)\neq\sigma_2(j)} & \text{, falls } i,j > 0 \\ d_{\sigma_1,\sigma_2}(i-2,j-2)+1 & \text{, falls } i,j > 1 \wedge \sigma_1(i) \neq \sigma_2(j-1) \\ & \quad \wedge \sigma_1(i-1) = \sigma_2(j) \end{cases}$$

Dabei bezeichnet **1** *die Indikatorfunktion [80] (*$\mathbf{1}_x$ *besitzt den Wert* 1*, falls die Bedingung x erfüllt ist, sonst* 0*).*
Dann heißt $Lev(\sigma_1,\sigma_2) := d_{\sigma_1,\sigma_2}(|\sigma_1|,|\sigma_2|)$ die ***Damerau-Levenshtein-Distanz*** *von σ_1 und σ_2.*

Zunächst wird eine kurze Interpretation der Damerau-Levenshtein-Distanz gegeben. Diese berechnet die minimale Anzahl einfügender, löschender und ersetzender Operationen, die benötigt werden, um eine Sequenz σ_1 in eine andere Sequenz σ_2 umzuwandeln. Dabei werden die Operationen Ersetzen, Einfügen und Löschen von Symbolen jeweils mit 1 gewichtet. Dies gilt auch für eine Vertauschung von zwei Symbolen. Bei der ursprünglichen Levenshtein-Distanz [81] werden Vertauschungen nicht direkt berücksichtigt. Deshalb wären eine Lösch- und eine Einfügeoperation notwendig, um eine Vertauschung zu realisieren. Folglich würde eine Vertauschung mit 2 gewichtet werden. Die Damerau-Levenshtein-Distanz berücksichtigt allerdings Vertauschungen und gewichtet diese ebenfalls mit 1. Natürlich kann auch in gewissen Anwendungsfällen eine Gewichtung von Vertauschungen mit einem anderen Wert $\neq 1$ sinnvoll sein. Dann kann das in diesem Abschnitt entwickelte Verfahren analog mit einer anderen Gewichtung verwendet werden.

Die Damerau-Levenshtein-Distanz erfüllt die mathematische Definition einer Metrik. Daraus lassen sich die folgenden Eigenschaften ableiten, die für den Sequenzvergleich von Bedeutung sein werden [78]:

- Die Damerau-Levenshtein-Distanz ist genau dann Null, wenn die beiden Sequenzen gleich sind: $Lev(\sigma_1,\sigma_2) = 0 \iff \sigma_1 = \sigma_2$
- Die Damerau-Levenshtein-Distanz beträgt maximal die Länge der längeren Sequenz: $Lev(\sigma_1,\sigma_2) \leq \max\{|\sigma_1|,|\sigma_2|\}$

So beträgt beispielsweise die Damerau-Levenshtein-Distanz der beiden Sequenzen $\sigma_1 = \langle ABCA \rangle$ und $\sigma_2 = \langle ACB \rangle$ 2, da σ_1 folgendermaßen mit Hilfe von zwei Operationen in σ_2 transformiert

werden kann:

1. $\langle ABCA \rangle$
2. $\langle ABC \rangle$ (Lösche das letzte Symbol A)
3. $\langle ACB \rangle$ (Tausche B mit C)

Um aus der Damerau-Levenshtein-Distanz ein Ähnlichkeitsmaß zu erhalten, wird sie auf den Wertebereich $[0, 1]$ skaliert, indem sie durch die Länge der längeren Sequenz geteilt wird. Des Weiteren wird die inverse Damerau-Levenshtein-Distanz berechnet, sodass ein höherer Wert der Damerau-Levenshtein-Distanz eine höhere Ähnlichkeit der Sequenzen impliziert:

$$Lev_{in}(\sigma_1, \sigma_2) := 1 - \frac{Lev(\sigma_1, \sigma_2)}{\max(|\sigma_1|, |\sigma_2|)}$$

Bisher wurde die Damerau-Levenshtein-Distanz nur für einzelne Sequenzen definiert. Um aber später Prozessmodelle miteinander vergleichen zu können, muss sie auf Mengen von Sequenzen erweitert werden. Analog zur *min-max*-Dichte (Kapitel 5.2) und *min-max*-Ähnlichkeit (Kapitel 5.4) werden dabei Sequenzen der Länge aus einem zuvor definierten Intervall $[n, m]$ betrachtet. Zur besseren Übersicht wird dafür zusätzlich eine abkürzende Schreibweise eingeführt:

Definition 28. *Für ein Prozessmodell P und $n, m \in \mathbb{N}_0$ mit $n \leq m$ sei*

$$\mathcal{S}_P^{n,m} := \bigcup_{i \in \{n,\ldots,m\}} P(i) = \{p_1, \ldots, p_{|\mathcal{S}_P^{n,m}|}\}.$$

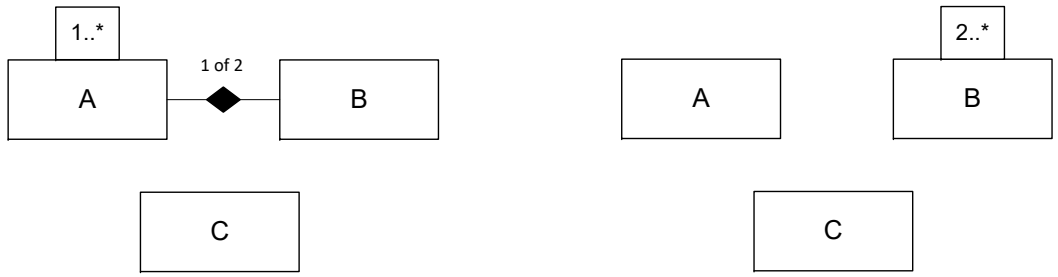
Die Menge $\mathcal{S}_P^{n,m}$ besteht also aus allen Sequenzen der Länge $\geq n$ und $\leq m$, die von P akzeptiert werden.

Um zwei Prozessmodelle P_1 und P_2 miteinander zu vergleichen, werden die korrespondierenden Mengen $\mathcal{S}_{P_1}^{n,m}$ und $\mathcal{S}_{P_2}^{n,m}$ mit Hilfe der Damerau-Levenshtein-Distanz verglichen. Dies geschieht durch ein im Prozessmanagement übliches Verfahren [82]: Jeder Sequenz $\sigma_1 \in \mathcal{S}_{P_1}^{n,m}$ von P_1 wird die bezüglich der Damerau-Levenshtein-Distanz ähnlichste Sequenz $\sigma_2 \in \mathcal{S}_{P_2}^{n,m}$ von P_2 zugeordnet und damit die Ähnlichkeit der Prozessmodelle berechnet [22]:

Definition 29. *Seien $n, m \in \mathbb{N}_{\geq 0}$ mit $n \leq m$ und Lev_{inv} die inverse und skalierte Damerau-Levenshtein-Distanz. Für zwei Prozessmodelle P_1 und P_2 heißt dann*

$$\Gamma_n^m(P_1, P_2) := \frac{\sum_{i=1}^{|\mathcal{S}_{P_1}^{n,m}|} \max\{Lev_{inv}(p_i, k) | k \in \mathcal{S}_{P_2}^{n,m}\}}{|\mathcal{S}_{P_1}^{n,m}|}$$

*die n-m-**Damerau-Levenshtein-Ähnlichkeit** von P_1 und P_2.*



(a) Prozessmodell Q_1 (b) Prozessmodell Q_2

Abbildung 5.2: Graphische ConDec-Repräsentation der Declare-Prozessmodelle Q_1 und Q_2 [22]

Es ist wichtig zu erwähnen, dass die n-m-Damerau-Levenshtein-Ähnlichkeit generell nicht symmetrisch ist, d.h. $\Gamma_n^m(P_1, P_2) \neq \Gamma_n^m(P_2, P_1)$. Dies kann folgendermaßen interpretiert werden: Der Aufwand, um Prozessmodell P_1 in Prozessmodell P_2 zu „transformieren“, kann sich vom Aufwand zur Transformation von Prozessmodell P_2 in Prozessmodell P_1 unterscheiden. Abhängig von der jeweiligen Anwendung kann $\Gamma_n^m(P_1, P_2)$, $\Gamma_n^m(P_2, P_1)$ oder beides ermittelt werden.

Zur Illustration der n-m-Damerau-Levenshtein-Ähnlichkeit werden einige Werte zu den folgenden Declare-Prozessmodellen Q_1 und Q_2 berechnet [22]: $Q_1 = (\{A, B, C\}, \mathcal{S}_1)$ und $Q_2 = (\{A, B, C\}, \mathcal{S}_2)$, wobei

$$\mathcal{S}_1 = \{\text{existence(A,1), exclusiveChoice(A,B)}\} \text{ und } \mathcal{S}_2 = \{\text{existence(B, 2)}\}.$$

Die graphische ConDec-Repräsentation der Prozessmodelle Q_1 und Q_2 ist in Abbildung 5.2 dargestellt. Es werden beispielhaft die 0-n-Damerau-Levenshtein-Ähnlichkeiten der beiden Prozessmodelle Q_1 und Q_2 für $n = 0, 1, 2, 3$ bestimmt. Dafür werden zunächst alle möglichen Sequenzen der entsprechenden Längen benötigt. Man erhält:

$$\mathcal{S}_{Q_1}^{0,0} = \emptyset = \mathcal{S}_{Q_2}^{0,0},$$

$$\mathcal{S}_{Q_1}^{0,1} = \{\langle A\rangle\},\ \mathcal{S}_{Q_2}^{0,1} = \emptyset,$$

$$\mathcal{S}_{Q_1}^{0,2} = \{\langle A\rangle, \langle AA\rangle, \langle AC\rangle, \langle CA\rangle\},\ \mathcal{S}_{Q_2}^{0,2} = \{\langle BB\rangle\},$$

$$\mathcal{S}_{Q_1}^{0,3} = \{\langle A\rangle, \langle AA\rangle, \langle AC\rangle, \langle CA\rangle, \langle AAA\rangle, \langle AAC\rangle, \langle ACA\rangle, \langle ACC\rangle, \langle CAA\rangle, \langle CAC\rangle, \langle CCA\rangle\},$$

$$\mathcal{S}_{Q_2}^{0,3} = \{\langle BB\rangle, \langle ABB\rangle, \langle BAB\rangle, \langle BBA\rangle, \langle BBB\rangle, \langle BBC\rangle, \langle BCB\rangle, \langle CBB\rangle\}.$$

Für $n = 0$ ist der Wert $\Gamma_0^0(Q_1, Q_2)$ nicht definiert, da beide Prozessmodelle keine Sequenz der Länge 0 akzeptieren. Ebenso lässt sich der Wert $\Gamma_0^1(Q_1, Q_2)$ für $n = 1$ nicht bestimmen, da Prozessmodell Q_2 keine Sequenz der Länge ≤ 1 akzeptiert.

Für $n = 2$ akzeptiert Prozessmodell Q_2 nur die Sequenz $\langle BB \rangle$. Für die Sequenz $\langle A \rangle$, die von Q_1 akzeptiert wird, werden zwei Operationen benötigt, um diese in $\langle BB \rangle$ zu transformieren: Das A muss durch B ersetzt werden und außerdem muss ein zusätzliches B eingefügt werden. Somit gilt $Lev(\langle A \rangle, \langle BB \rangle) = 2$. Da alle Sequenzen der Länge 2, die von Q_1 akzeptiert werden $(\langle AA \rangle, \langle AC \rangle, \langle CA \rangle)$, kein B beinhalten, sind ebenso zwei Operationen (Ersetzungen) notwendig, um Sequenz $\langle BB \rangle$ zu erhalten. Somit gilt $Lev(\sigma, \langle BB \rangle) = 2$ für alle $\sigma \in \mathcal{S}_{Q_1}^{0,2}$. Dies impliziert eine 0-2-Damerau-Levenshtein-Ähnlichkeit $\Gamma_0^2(Q_1, Q_2)$ von 0.

Für $n = 3$ sind die Werte der Damerau-Levenshtein-Distanz in Tabelle 5.4 dargestellt. Man sieht, dass entweder zwei oder drei Operationen notwendig sind, um eine Sequenz von Q_1 in eine Sequenz von Q_2 zu transformieren. Um die 0-3-Damerau-Levenshtein-Ähnlichkeit zu bestimmen, werden die Werte der normierten, inversen Damerau-Levenshtein-Distanz benötigt. Diese sind in Tabelle 5.5 abgebildet. Es ist offensichtlich, dass alle Einträge, bei denen die Damerau-Levenshtein-Distanz 3 ist, eine normierte Damerau-Levenshtein-Distanz von 0 besitzen, da 3 das Maximum beträgt. Für alle Werte mit Damerau-Levenshtein-Distanz 2 ergibt sich $Lev_{inv} = 1 - \frac{2}{3} = \frac{1}{3} = 0,33$. Nun kann die 0-3-Damerau-Levenshtein-Ähnlichkeit berechnet werden:

$$\Gamma_0^3(Q_1, Q_2) = \frac{\sum_{i=1}^{|\mathcal{S}_{P_1}^{0,3}|} \max\{Lev_{inv}(p_i, k) | k \in \mathcal{S}_{Q_2}^{0,3}\}}{|\mathcal{S}_{Q_1}^{0,3}|} = \frac{\sum_{i=1}^{11} \max\{Lev_{inv}(p_i, k) | k \in \mathcal{S}_{Q_2}^{0,3}\}}{11}$$

Für jede Sequenz $\sigma \in \mathcal{S}_{P_1}^{0,3}$ beträgt der größte Eintrag in Tabelle 5.5 den Wert 0.33. Folglich gilt, dass $\max\{Lev_{inv}(\sigma, k) | k \in \mathcal{S}_{Q_2}^{0,3}\} = 0,33$. Damit ergibt sich:

$$\frac{\sum_{i=1}^{11} \max\{Lev_{inv}(p_i, k) | k \in \mathcal{S}_{Q_2}^{0,3}\}}{11} = \frac{\sum_{i=1}^{11} 0,33}{11} = \frac{11 \cdot 0,33}{11} = 0,33 = 33\%$$

Insgesamt beträgt die 0-3-Damerau-Levenshtein-Ähnlichkeit $\Gamma_0^3(Q_1, Q_2)$ also 33%. Analog können weitere Damerau-Levenshtein-Ähnlichkeiten berechnet werden. Tabelle 5.6 zeigt alle Werte der 0-n-Damerau-Levenshtein-Ähnlichkeiten $\Gamma_0^n(Q_1, Q_2)$ und $\Gamma_0^n(Q_2, Q_1)$ bis $n = 7$. Alle Resultate in Tabelle 5.6 werden in Kapitel 8 erläutert, diskutiert und interpretiert.

	$\langle ABB \rangle$	$\langle BAB \rangle$	$\langle BB \rangle$	$\langle BBA \rangle$	$\langle BBB \rangle$	$\langle BBC \rangle$	$\langle BCB \rangle$	$\langle CBB \rangle$
$\langle A \rangle$	2	2	2	2	3	3	3	3
$\langle AA \rangle$	2	2	2	2	3	3	3	3
$\langle AAA \rangle$	2	2	3	2	3	3	3	3
$\langle AAC \rangle$	2	2	3	3	3	2	3	3
$\langle AC \rangle$	2	2	2	3	3	2	2	3
$\langle ACA \rangle$	2	3	3	2	3	3	2	3
$\langle ACC \rangle$	2	3	3	3	3	2	2	3
$\langle CA \rangle$	3	2	2	2	3	3	2	2
$\langle CAA \rangle$	3	2	3	2	3	3	3	2
$\langle CAC \rangle$	3	2	3	3	3	2	3	2
$\langle CCA \rangle$	3	3	3	2	3	3	2	2

Tabelle 5.4: Werte der Damerau-Levenshtein-Distanz für $\mathcal{S}_{Q_1}^{0,3}$ und $\mathcal{S}_{Q_2}^{0,3}$

	$\langle ABB \rangle$	$\langle BAB \rangle$	$\langle BB \rangle$	$\langle BBA \rangle$	$\langle BBB \rangle$	$\langle BBC \rangle$	$\langle BCB \rangle$	$\langle CBB \rangle$
$\langle A \rangle$	0,33	0,33	0,33	0,33	0	0	0	0
$\langle AA \rangle$	0,33	0,33	0,33	0,33	0	0	0	0
$\langle AAA \rangle$	0,33	0,33	0	0,33	0	0	0	0
$\langle AAC \rangle$	0,33	0,33	0	0	0	0,33	0	0
$\langle AC \rangle$	0,33	0,33	0,33	0	0	0,33	0,33	0
$\langle ACA \rangle$	0,33	0	0	0,33	0	0	0,33	0
$\langle ACC \rangle$	0,33	0	0	0	0	0,33	0,33	0
$\langle CA \rangle$	0	0,33	0,33	0,33	0	0	0,33	0,33
$\langle CAA \rangle$	0	0,33	0	0,33	0	0	0	0,33
$\langle CAC \rangle$	0	0,33	0	0	0	0,33	0	0,33
$\langle CCA \rangle$	0	0	0	0,33	0	0	0,33	0,33

Tabelle 5.5: Werte der normierten, inversen Damerau-Levenshtein-Distanz für $\mathcal{S}_{Q_1}^{0,3}$ und $\mathcal{S}_{Q_2}^{0,3}$

n	0	1	2	3	4	5	6	7
$\Gamma_0^n(Q_1, Q_2)$	-	-	0	0,33	0,5	0,6	0,67	0,71
$\Gamma_0^n(Q_2, Q_1)$	-	-	0	0,29	0,43	0,51	0,56	0,6

Tabelle 5.6: 0-n-Damerau-Levenshtein-Ähnlichkeiten $\Gamma_0^n(Q_1, Q_2)$ und $\Gamma_0^n(Q_2, Q_1)$ bis $n = 7$ [22]

5.6 Ähnlichkeit von Prozessmodellen in verwandten Arbeiten

Grundsätzlich werden in der Literatur zur Bestimmung von Ähnlichkeiten von Prozessmodellen drei verschiedene Arten von Ähnlichkeiten unterschieden [69]:

- Die *Ähnlichkeit der beteiligten Elemente* (engl. *node matching similarity*) beschreibt die Ähnlichkeit der Bezeichnungen aller am Prozess beteiligten Elemente, z.B. Aktivitäten oder

Organisationen.

- Die *strukturelle Ähnlichkeit* (engl. *structural similarity*) beschreibt die Ähnlichkeit bezüglich der Strukturen der Prozessmodelle. Diese werden beispielsweise mit Hilfe des *Graph-Editierabstands* (engl. *graph edit distance*) [83] bestimmt.

- Die *verhaltensorientierte Ähnlichkeit* (engl. *behavioral similarity*) untersucht Prozessmodelle auf Ähnlichkeiten bezüglich ihres jeweiligen Verhaltens, d.h. deren Sprachen.

Analog zur Diskussion von Vergleichsmethoden von Prozessmodellen (Kapitel 3.4) wird sich im Folgenden auf die *verhaltensorientierte Ähnlichkeit* fokussiert. Alle Argumente, die bereits in Kapitel 3.4 zum Vergleich der *natürlichen Sprachdimension* und der *Graphdimension* diskutiert wurden, können hier gleichermaßen auf die *Ähnlichkeit der beteiligten Elemente* und die *strukturelle Ähnlichkeit* angewandt werden.

Für die *verhaltensorientierte Ähnlichkeit* existieren Ansätze, die die Komplexitäten der Sprachen der jeweiligen Prozessmodelle miteinander vergleichen [67]. In realen Anwendungen sind aber vielmehr konkrete Ausführungen anstatt der Komplexitäten der Sprachklassen von Relevanz. Weitere Untersuchungen werden mit sogenannten *kausalen Fußabdrücken* (engl. *causal footprints*) [84] unternommen, deren Verwendung allerdings nur eine Annäherung und nicht das exakte Verhalten der Prozessmodelle abdeckt [69]. Der Ansatz in [85] berechnet Ähnlichkeiten von Prozessmodellen in Bezug auf endliche Mengen von Sequenzen, die durch Simulationen der Modelle entstehen. Dabei werden aber im Regelfall nicht alle möglichen Sequenzen abgedeckt [69].

Insgesamt lässt sich feststellen, dass trotz zahlreicher Ansätze in der Literatur nach wie vor die Notwendigkeit nachvollziehbarer und anwendbarer Ähnlichkeitsansätze besteht. Die in dieser Arbeit entwickelten Ähnlichkeitsmaße (siehe Kapitel 5) widmen sich dieser offenen Herausforderung. Bei mehreren durchgeführten Berechnungen und Evaluationen (Kapitel 8.3) zeigt sich zudem ihr praktischer Nutzen in konkreten Anwendungsfällen.

Kapitel 6

Szenario-basierte Modellprüfung

Beim Entwurf und der Weiterentwicklung von Prozessmodellen steht die Prozessmodelliererin oder der Prozessmodellierer regelmäßig vor der Herausforderung, die gegenwärtige Korrektheit des Prozessmodells zu gewährleisten. Während des Verifizierungsprozesses, beispielsweise bei der Überprüfung, ob die beabsichtigten Modifikationen korrekt umgesetzt wurden, ergeben sich häufig komplexe Fragestellungen, die sich auf spezifische Ausführungssequenzen beziehen [18, 25]:

1. Spiegelt der aktuelle Zustand der Prozessausführung eine in der Realität vorkommende Situation wider?
2. Welche sind alle validen Fortsetzungen der aktuell laufenden Prozessinstanz?
3. Welche Aktivität darf im nächsten Schritt durchgeführt werden?
4. Kann eine gewisse (bisher ungültige, da noch unvollständige) Teilsequenz innerhalb von n Schritten zu einer gültigen Sequenz fortgesetzt werden?

Gerade im Bereich der deklarativen Prozessmodellierung sind derartige Fragen im Allgemeinen gar nicht oder nur sehr schwer zu beantworten [18, 25, 86]. Fragestellungen dieser Art werden in der vorliegenden Arbeit als *Szenarien* bezeichnet.

In diesem Kapitel werden drei Verhaltensanalysen (Kapitel 6.1-6.3) definiert und beschrieben [18]. Mit deren Hilfe können Lösungen zu obig beschriebenen Szenarien berechnet werden. Abschließend wird in Abschnitt 6.4 ein Überblick über die Literatur, die sich mit der Verständlichkeit deklarativer Prozessmodelle beschäftigt, gegeben.

In den folgenden Unterkapiteln wird mit $M_P = (\mathcal{A}, Q, q_0, \delta, F)$ der Prozessautomat (Kapitel 3.2.1) zu einem Prozessmodell P über einer endlichen Menge von Aktivitäten $\mathcal{A}$ bezeichnet.

6.1 Verhaltensanalyse 1: Gültigkeit von Sequenzen

Eine häufige Aufgabe bei der Validierung von Prozessmodellen ist es, gewisse Sequenzen auf Gültigkeit zu überprüfen. Damit wird es ermöglicht, wichtige Eigenschaften von Prozessmodellen zu validieren und/oder nicht-gewolltes Verhalten (d.h. im Prozessmodell valide Sequenzen, die eigentlich nicht erlaubt sein sollten) zu identifizieren.

Verhaltensanalyse 1 beschreibt das Verifizieren von Sequenzen auf Gültigkeit [18]. Zu ihrer Umsetzung wird untersucht, ob der zu einem Prozessmodell zugehörige Prozessautomat $M_P = (\mathcal{A}, Q, q_0, \delta, F)$ die jeweilige Sequenz t akzeptiert. Dazu muss lediglich überprüft werden, ob das Lesen von t in einem akzeptierenden Zustand endet, d.h. ob $\sigma(t) \in F$. Das zugehörige Verfahren ist in Algorithmus 7 dargestellt.

Algorithmus 7: *checkTrace*

Input: Process Automaton $M_P = (\mathcal{A}, Q, q_0, \delta, F)$, trace t
Output: True, if M_P accepts t, otherwise False

```
1 if δ(q0, t) ∈ F then
2 |   return True
3 else
4 |   return False
5 end
```

Im Falle von deklarativen Prozessmodellen ist natürlich die Frage (für eine ungültige Sequenz t) nach den verletzten Constraints relevant. Diese Frage kann ebenso mit Hilfe von Algorithmus 7 beantwortet werden. Dazu wird zu jedem Constraint $\tau \in \mathcal{T}$ des deklarativen Prozessmodells $P = (\mathcal{A}, \mathcal{T})$ ein eigenes Prozessmodell $P_\tau = (\mathcal{A}, \{\tau\})$ erstellt und der zugehörige Prozessautomat M_{P_τ} berechnet [18]. Anschließend wird auf jedem dieser Prozessautomaten M_{P_τ} Algorithmus 7 aufgerufen. Falls das Ergebnis *True* ist, erfüllt t das jeweilige Constraint τ. Andernfalls (Ergebnis *False*) ist τ verletzt. Algorithmus 8 zeigt das gesamte Vorgehen. Diesem werden ein deklaratives Prozessmodell $P = (\mathcal{A}, \mathcal{T})$ und eine Sequenz t übergeben. In der for-Schleife wird für jedes Constraint $\tau \in \mathcal{T}$ der zugehörige Prozessautomat berechnet (Zeile 3). Anschließend wird die Sequenz t in diesem Prozessautomaten auf Akzeptanz geprüft (Zeile 4). Falls dieser Prozessautomat die Sequenz t nicht akzeptiert, wird das korrespondierende Constraint τ zur Ausgabe hinzugefügt (Zeile 5). Abschließend werden alle derartig hinzugefügten Constraints zurückgegeben (Zeile 8).

Algorithmus 8: *findViolatedConstraints*

Input: Declare model $P = (\mathcal{A}, \mathcal{T})$, trace t
Output: set S of violated constraints

```
1 S ← ∅
2 for τ ∈ T do
3 |   M_Pτ ← calculateProcessAutomaton((A, {τ}))
4 |   if checkTrace(M_Pτ, t) == False then
5 |   |   S.add(τ)
6 |   end
7 end
8 return S
```

6.2 Verhaltensanalyse 2: Vervollständigung von Sequenzen

Im vorherigen Kapitel wird eine Sequenz t auf Gültigkeit bezüglich eines Prozessmodells P geprüft. Falls eine Sequenz ungültig ist, bedeutet dies aber nicht, dass sie keine gültige Sequenz mehr werden kann. Durch die Ausführungen weiterer Aktivitäten kann eine zum aktuellen Zeitpunkt ungültige Sequenz t dennoch noch „gerettet“, d.h. zu einer gültigen Sequenz fortgesetzt werden.

Als erstes Beispiel sei dazu das Declare-Modell $P_1 = (\{A, B\}, \{\text{response}(A, B)\})$, das aus zwei Aktivitäten A und B sowie einem response Constraint besteht, gegeben. Das Überprüfen der Sequenz $\langle A \rangle$ auf Gültigkeit (Algorithmus 8) liefert *False*, da nach der Ausführung von Aktivität A keine Ausführung von Aktivität B folgt. Allerdings kann t zu einer gültigen Sequenz fortgesetzt werden, da bereits durch das Anhängen von Aktivität B eine gültige Sequenz, nämlich $\langle AB \rangle$, entsteht. Außerdem wären etwa auch die Sequenzen $\langle AAB \rangle, \langle ABB \rangle, \langle AAAB \rangle$ oder $\langle AABB \rangle$ valide Fortsetzungen. In diesem Fall gibt es sogar unendlich viele gültige Vervollständigungen von $\langle A \rangle$, da beispielsweise jede Sequenz der Form $\langle A \dots AB \rangle$ bestehend aus beliebig vielen As und einem B am Ende eine gültige Vervollständigung ist.

Als zweites Beispiel wird das Prozessmodell $P_2 = (\{A, B\}, \{\text{precedence}(B, A)\})$, bestehend ebenfalls aus zwei Aktivitäten A und B sowie einem precedence Constraint, betrachtet. Analog zu P_1 liefert das Überprüfen der Sequenz $\langle A \rangle$ *False*, da das precedence Constraint nicht erfüllt ist (vor der Ausführung von Aktivität A hätte Aktivität B ausgeführt werden müssen). Im Gegensatz zu P_1 kann die Sequenz $\langle A \rangle$ bezüglich P_2 allerdings nicht zu einer gültigen Sequenz fortgesetzt werden, da das precedence Constraint nicht mehr erfüllt werden kann (Aktivität B müsste vor Aktivität A ausgeführt werden).

In der Folge wird eine Methode vorgestellt, die es ermöglicht zu überprüfen, ob (aktuell ungültige) Sequenzen bezüglich eines Prozessmodells noch zu einer gültigen Sequenz vervollständigt werden können [18]. Wenn dies der Fall ist, wird eine minimale gültige Ver-

vollständigung, d.h. eine gültige Vervollständigung minimaler Länge, berechnet.

Zur Beantwortung von Verhaltensanalyse 2 wird die Minimalitätseigenschaft des Prozessautomaten $M_P = (\mathcal{A}, Q, q_0, \delta, F)$ (Kapitel 2.2.7) verwendet, auf Grund derer es genau einen „Totzustand“ $q_d \in Q$ gibt, sodass [50]:

1. $q_d \notin F$
2. $\forall a \in \mathcal{A} : \delta(q_d, a) = q_d$

Das bedeutet, dass dieser Zustand q_d ein nicht-akzeptierender Zustand (Punkt 1) ist, der nicht mehr verlassen werden kann (Punkt 2). Insbesondere können alle Wörter, die in den Zustand q_d führen, nicht mehr zu einem gültigen Wort vervollständigt werden, da q_d nicht mehr verlassen werden kann und demzufolge auch kein akzeptierender Zustand mehr erreicht werden kann. Aus Prozesssicht gesprochen bedeutet dies, dass sich die bisherige Teilsequenz zu keiner validen Sequenz fortsetzen lässt [18].

Andererseits gilt auf Grund der Minimalitätseigenschaft, dass es von allen anderen Zuständen $q \in Q \setminus \{q_d\}$ einen Pfad in einen akzeptierenden Zustand geben muss und dass das bis dahin verarbeitete Wort auf jeden Fall zu einem gültigen Wort fortgesetzt werden kann. Darum ergibt sich für die Vervollständigung einer Sequenz t die folgende Bedingung:

$$t \text{ kann valide vervollständigt werden} \iff \delta(q_0, t) \neq q_d$$

Im Falle, dass $\sigma(q_0, t) \neq q_d$ gilt, soll eine minimale gültige Vervollständigung von t berechnet werden. Dafür wird der Prozessautomat $M_P = (\mathcal{A}, Q, q_0, \delta, F)$ als gewichteter Graph $G = (V, E)$ [87], dessen Knoten V aus der Menge Q der Zustände von M_P und die Kanten E aus den Transitionen (assoziiert mit den Übergangssymbolen als Gewichte) von M_P bestehen, betrachtet. Dies ermöglicht das Anwenden klassischer Algorithmen aus der Graphentheorie wie z.B. die *Breitensuche* [75], die nun angewandt wird, um einen kürzesten Pfad in einen akzeptierenden Zustand zu suchen. Der Algorithmus bricht ab, sobald ein akzeptierender Zustand erreicht wird und speichert alle Pfade kürzester Länge in einen akzeptierenden Zustand. Mit Hilfe der kürzesten Pfades können nun alle Vervollständigungen minimaler Länge konstruiert werden, indem alle Wörter entlang der Pfade konstruiert werden.

Beispielhaft wird der in Abbildung 6.1 illustrierte Graph G betrachtet. Dabei ist q_0 der Startzustand und q_2 der einzige akzeptierende Zustand. Möchte man das leere Wort ε vervollständigen, würde die Breitensuche den kürzesten Pfad von q_0 in einen akzeptierenden Pfad suchen und folglich nur einen Pfad, nämlich $q_0 - q_1 - q_2$, zurückgeben. Auf diesem Pfad können zwei verschiedene Wörter konstruiert werden, da auf der Transition von q_0 nach q_1 zwei Symbole (A und B) und auf der Transition von q_1 nach q_2 nur ein Symbol, nämlich C, stehen. Somit werden $\langle AC \rangle$ und $\langle BC \rangle$ als minimale Vervollständigungen berechnet.

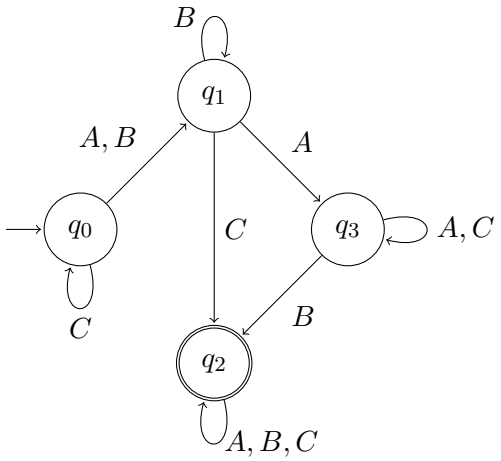


Abbildung 6.1: Beispielgraph G

Das komplette Verfahren zur Berechnung minimaler Vervollständigungen ist in Algorithmus 9 dargestellt. Der Input des Algorithmus besteht aus einem Prozessautomaten P_M und einer zu vervollständigenden (Teil-)Sequenz t. Zunächst (Zeile 1) wird überprüft, ob man durch Auswerten von t in P_M im Totzustand q_d landet. Wenn dies der Fall ist, kann t nicht mehr valide vervollständigt werden und der Algorithmus liefert *False* zurück (Zeile 6). Andernfalls kann t valide vervollständigt werden und es werden alle kürzesten Pfade in einen akzeptierenden Zustand mit Hilfe einer Breitensuche berechnet (Zeile 2). Anschließend werden alle möglichen Sequenzen entlang der gefunden Pfade konstruiert (Zeile 3) und zurückgegeben (Zeile 4).

Algorithmus 9: *checkCompletion*

Input: Process Automaton $P_M = (\mathcal{A}, Q, q_0, \delta, F)$, trace t
Output: Set C of completed traces of minimal length, if t can be completed successfully, otherwise False

```
1 if δ(q_0, t) ≠ q_d then
2 |   Γ_{δ(q_0,t)} ←breadth-first_search(δ(q_0, t))
3 |   S ← constructWords(Γ_{δ(q_0,t)})
4 |   return S
5 else
6 |   return False
7 end
```

6.3 Verhaltensanalyse 3: Vervollständigung von Sequenzen innerhalb von n Schritten

Im vorherigen Unterkapitel werden Sequenzen auf mögliche Vervollständigung überprüft, d.h. es wird verifiziert, ob sie zu einer gültigen Prozessinstanz fortgesetzt werden können. Diese Fragestellung wird in diesem Kapitel präzisiert, indem geprüft wird, ob Sequenzen innerhalb von n Schritten für festes $n \in \mathbb{N}$ erfolgreich beendet werden können [18]. Dafür werden zunächst als Beispiel das Declare-Modell

$$P = (\{A, B, C\}, \{\text{response}(A, B), \text{response}(A, C)\})$$

und die Sequenz $t = \langle A \rangle$ betrachtet. Durch die beiden response Constraints wird durch die Ausführung von Aktivität A die Ausführung der beiden Aktivitäten B und C gefordert. Da keine weiteren Constraints existieren, liefert die Überprüfung von t auf Vervollständigung (Algorithmus 9) mit $\langle ABC \rangle$ und $\langle ACB \rangle$ die einzigen kürzesten validen Vervollständigungen von t. Folglich kann t nicht innerhalb von $n = 1$ Schritt vervollständigt werden, wohl aber in $n = 2$ Schritten. Des Weiteren gilt, dass t in n Schritten für alle $n \geq 2$ vervollständigt werden kann, da beispielsweise an die Sequenz $\langle ABC \rangle$ beliebig viele Bs oder Cs angefügt werden können, ohne dass dabei das Prozessmodell P verletzt wird.

Zur Beantwortung der Frage nach der Vervollständigung einer Sequenz σ in n Schritten wird, wie in Kapitel 6.2, eine Breitensuche [75] durchgeführt. Im Gegensatz zu Verhaltensanalyse 2, bei der mit Hilfe der Breitensuche der nächste akzeptierende Zustand gesucht wurde, wird hier allerdings eine Breitensuche der Breite n durchgeführt [18]. Diese liefert alle Pfade der Länge $\leq n$ ab dem aktuellen Zustand $\delta(q_0, t)$ in einen akzeptierenden Zustand $q \in F$. Mit Hilfe dieser Pfade können analog zu Kapitel 6.2 alle gültigen Sequenzen bis zur Länge $|t| + n$ konstruiert werden.

Als Beispiel werden nun der Graph G' in Abbildung 6.2 sowie das leere Wort ε betrachtet. Für $n = 0$ gibt es keinen Pfad ab q_0 in den einzigen akzeptierenden Zustand q_2. Für $n = 1$ existiert ein Pfad, nämlich $q_0 - q_2$. Als minimale Vervollständigung wird somit $\langle C \rangle$ berechnet. Für $n = 2$ erhält man die zwei unterschiedlichen Pfade $q_0 - q_1 - q_2$ und $q_0 - q_2 - q_2$. Diese liefern wiederum die Vervollständigungen $\langle AC \rangle, \langle BC \rangle, \langle CA \rangle, \langle CB \rangle$ und $\langle CC \rangle$. Tabelle 6.1 zeigt alle Pfade und die zugehörigen Vervollständigungen bis $n = 3$.

Das komplette Verfahren zur Berechnung von Vervollständigungen der Länge n ist in Algorithmus 10 zusammengefasst. Dabei werden als Input ein Prozessautomat P_M, eine zu vervollständigende (Teil-)Sequenz t und eine Länge n benötigt. Es wird eine Breitensuche der Breite n beginnend im Zustand $\delta(q_0, t)$ aufgerufen (Zeile 1). Diese liefert alle Pfade der Länge $\leq |t| + n$ von $\delta(q_0, t)$ in einen akzeptierenden Zustand zurück. Abschließend werden anhand der gefundenen Pfade alle möglichen Fortsetzungen berechnet (Zeile 2) und zurückgegeben (Zeile 3).

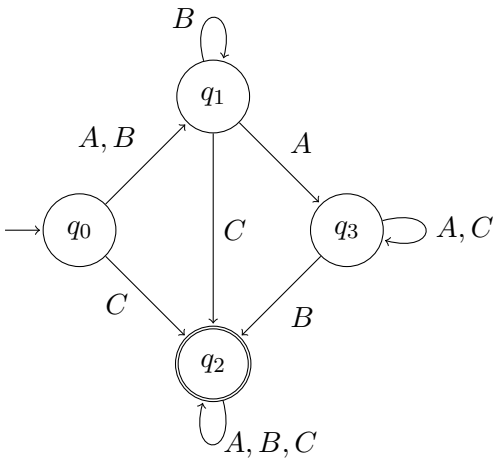


Abbildung 6.2: Beispielgraph G'

n	Pfade	Vervollständigungen
0	$\emptyset$	$\emptyset$
1	$q_0 - q_2$	$\langle C \rangle$
2	$q_0 - q_1 - q_2$	$\langle AC \rangle, \langle BC \rangle$
	$q_0 - q_2 - q_2$	$\langle CA \rangle, \langle CB \rangle, \langle CC \rangle$
3	$q_0 - q_1 - q_3 - q_2$	$\langle AAB \rangle, \langle BAB \rangle$
	$q_0 - q_1 - q_1 - q_2$	$\langle ABC \rangle, \langle BBC \rangle$
	$q_0 - q_1 - q_2 - q_2$	$\langle ACA \rangle, \langle ACB \rangle, \langle ACC \rangle, \langle BCA \rangle, \langle BCB \rangle, \langle BCC \rangle$
	$q_0 - q_2 - q_2 - q_2$	$\langle CAA \rangle, \langle CAB \rangle, \langle CAC \rangle, \langle CBA \rangle, \langle CBB \rangle, \langle CBC \rangle, \langle CCA \rangle, \langle CCB \rangle, \langle CCC \rangle$

Tabelle 6.1: Alle validen Vervollständigungen von G ab Zustand q_0 bis zur Länge $n = 3$

Algorithmus 10: *checkCompletionNSteps*

Input: Process Automaton $P_M = (\mathcal{A}, Q, q_0, \delta, F)$, trace t, number of steps n
Output: Set S of valid traces of length $\leq |t| + n$

```
1 P_{δ(q0,t)} ←breadth-first_search(δ(q0, t), n)
2 S ← constructWords(P_{δ(q0,t)})
3 return S
```

6.4 Verständlichkeit deklarativer Prozessmodelle in verwandten Arbeiten

Trotz der weithin bekannten Vorteile von deklarativen Prozessmodellierungssprachen für sogenannte flexible Prozesse [88] erweisen sich die entsprechenden deklarativen Prozessmodelle, die im Allgemeinen aus einer Vielzahl von Constraints bestehen, als für Menschen kompliziert und schwer zu verstehen [17, 89]. Dies liegt hauptsächlich an der gegenseitigen Wechselwirkung der Constraints [17], die es selbst Fachleuten äußerst schwer macht, das gesamte Prozessmodell zu überblicken und zu verstehen, ohne dabei wichtige Einzelheiten zu übersehen. Eine Studie in [17] ergab, dass Personen die Kombination von grafischen und textbasierten Elementen nutzen, um deklarative Modelle, die in Declare [6] modelliert sind, zu interpretieren. In [90] zeigt eine weitere Studie, dass die Verwendung von verschiedenen Farben das intuitive Verständnis von DCR-Graphen erleichtern kann. Weitere Ansätze hybrider Prozessmodellierungssprachen, die sowohl grafische als auch textbasierte Elemente vereinen, haben diese Herausforderung nicht ausreichend bewältigt und führten zu ähnlichen Schwierigkeiten [91, 92].

Aus obigen Gründen wurden in letzter Zeit Ansätze entwickelt, die darauf abzielen, deklarative Prozessmodelle zu erforschen und Menschen dabei zu unterstützen, sie besser zu verstehen. So beschäftigen sich [93, 19] mit der Identifizierung sogenannter „versteckter Abhängigkeiten“ (engl. *hidden dependencies*) in deklarativen Prozessmodellen. Versteckte Abhängigkeiten entstehen durch das Zusammenspiel mehrerer Constraints und sind daher nicht explizit im Modell dargestellt, was es für Menschen schwierig macht, sie zu erkennen. Diese erkannten versteckten Abhängigkeiten werden den Modellen in Form von grafischen und textlichen Annotationen hinzugefügt, um ihre Verständlichkeit zu verbessern [93]. Darüber hinaus untersuchen [15, 94] die Menge der Constraints in deklarativen Prozessmodellen auf Redundanzen und „Inkonsistenzen“, die dazu führen, dass keine einzige Sequenz erfüllt werden kann und das Prozessmodell somit unbrauchbar wird. [71] schlägt eine mögliche Lösung zur Behebung dieser erkannten Inkonsistenzen vor. Alle diese Ansätze tragen dazu bei, deklarative Prozessmodelle verständlicher zu gestalten. Dennoch bieten sie keine Unterstützung für konkrete Szenarien zur Laufzeit, wie etwa die Frage: „Welche möglichen Fortsetzungen gibt es für eine aktuell laufende Prozessinstanz?“.

In [95] wird eine Laufzeitverifizierungstechnik für lineare temporale Logik (LTL) vorgestellt. Diese ermöglicht die Überprüfung von Sequenzen auf Erfüllung von Bedingungen. Da deklarative Prozessmodellierungssprachen wie Declare [6] auf linearer temporaler Logik über endlichen Sequenzen (LTL_f) basieren [96], kann dieser Ansatz dazu beitragen, Declare-Modelle zur Laufzeit zu verifizieren. Dennoch können solche konkreten Szenarien (z.B. *„Welche sind alle validen Fortsetzungen der aktuell laufenden Prozessinstanz?“*), wie bereits erwähnt, nicht adäquat behandelt werden.

Dennoch bleibt festzuhalten, dass trotz dieser Fortschritte deklarative Prozessmodelle nach wie vor oft als sehr unverständlich wahrgenommen werden, insbesondere in Bezug auf die komplexe Interaktion der Constraints. Die in dieser Dissertation vorgestellten Verhaltensanalysen bieten einen bedeutenden Beitrag zur Verbesserung der Verständlichkeit solcher Modelle und tragen dazu bei, die Herausforderungen bei der Interpretation und Analyse von deklarativen Prozessmodellen zu bewältigen (Kapitel 8.6).

Kapitel 7

Implementierung

Im Rahmen dieser Dissertation wurden die erarbeiteten Methoden und Konzepte für Evaluations- und Illustrationszwecke programmatisch in Java implementiert. Dabei entstanden zwei Applikationen, die im weiteren Verlauf genauer erläutert werden. Das erste entwickelte Werkzeug ist eine Konsolenanwendung, die den automatisierten Vergleich sowie weitere in Kapitel 3 entwickelte Analyseverfahren für imperative und deklarative Prozessmodelle ermöglicht. Mithilfe dieser Implementierung werden in den Kapiteln 8.2, 8.3 und 8.5 die in dieser Arbeit entwickelten Ansätze zum Modellvergleich, zur Berechnung von Ähnlichkeiten und von Verhaltensänderungen auf realen Datensätzen evaluiert. Hierfür werden in Abschnitt 7.1 alle notwendigen Syntaxdefinitionen eingeführt. Ein umfassender Überblick über die Funktionalitäten und Anwendungen des entwickelten Werkzeugs wird dann in Kapitel 7.2 gegeben. Abschließend beschreibt Kapitel 7.3 eine entwickelte Browseranwendung zur szenario-basierten Modellprüfung aus Kapitel 6, die in Abschnitt 8.6 die Grundlage für eine durchgeführte Benutzerstudie bildet.

7.1 Syntax-Definitionen

Die entwickelten Applikationen verwenden eine eigene Syntax für Declare-Modelle, BPMN-Modelle sowie endliche deterministische Automaten, um die Verarbeitung von Modellen zu erleichtern. Dabei werden jedoch keine signifikanten Einschränkungen vorgenommen, sodass sämtliche Funktionalitäten der verwendeten Konstrukte in die Implementierung integriert sind. Diese werden benötigt, um Modelle bzw. Automaten den Applikationen übergeben zu können und werden im Folgenden beschrieben.

7.1.1 Syntax für BPMN-Modelle

BPMN verfügt über eine eigene Modellsyntax mit der Dateiendung „.bpmn“, die auf der *Extensible Markup Language (XML)*[14] basiert. Die vorliegende Applikation verwendet allerdings eine eigene Syntax für BPMN-Modelle. Dabei werden BPMN-Modelle in Form von *Java Script Object Notation (JSON)*-Dateien angenommen, in denen die Flussobjekte und Sequenzflüsse des Prozessmodells definiert sind.

Zur Definition der Menge der Aktivitäten von BPMN-Modellen werden Tupel der Form $[A, B]$ akzeptiert. A ist ein beliebiger String, der den Namen der Aktivität repräsentiert. B ist dabei ein einstelliges Zeichen, das zur eindeutigen Identifizierung des Knotens innerhalb des Modells und somit als eine Art ID dient. Aktivitäten mit demselben Zeichen B werden als äquivalent hinsichtlich der Aktivitätssequenz-Entscheidung betrachtet. Die Sequenzflüsse werden als eine Menge von Tupeln $[A, B]$ definiert, wobei A den Ausgangspunkt und B den Endpunkt eines Sequenzflusses repräsentiert. Hierbei ist darauf zu achten, dass die beiden Flussobjekte A und B bereits in einer anderen Menge definiert sein müssen. Andernfalls ist die Syntax verletzt. Falls es sich dabei um eine Aktivität handelt, muss die ID der Aktivität verwendet werden. Die Definition der Events (aufgeteilt in Start- und Endevents) und Gateways (aufgeteilt in exklusive, inklusive und parallele Gateways) erfolgt jeweils durch die Vergabe eindeutiger Bezeichner. Formal sieht die Syntax für BPMN-Modelle in Backus-Naur-Form (BNF) [97] folgendermaßen aus:

```
<BPMN-Modell> ::= "{"
                    "Activities:" "[" <Aktivitäten> "],"
                    "StartEvent:" <Ereignis> ","
                    "EndEvents:" "[" <EreignisListe> "],"
                    "ExclusiveGateways:" "[" <GatewayListe> "],"
                    "InclusiveGateways:" "[" <GatewayListe> "],"
                    "ParallelGateways:" "[" <GatewayListe> "],"
                    "SequenceFlows:" "[" <SequenceFlowListe> "]"
                  "}"
<Aktivitäten> ::= <Aktivität> | <Aktivität> "," <Aktivitäten>
<Aktivität> ::= "[" <AktivitätsName> "," <AktivitätsID> "]"
<AktivitätsName> ::= Name der Aktivität
<AktivitätsID> ::= ID der Aktivität
<Ereignis> ::= Bezeichner des Ereignisses
<EreignisListe> ::= <Ereignis> | <Ereignis> "," <Ereignisliste>
<Gateway> ::= Bezeichner des Gateways
```

[14] https://www.w3.org/TR/xml/, abgerufen am: 27. Dezember 2023

```
{
"Activities": [["A","A"],["B","B"],["C","C"]],
"StartEvent": "s",
"EndEvents": ["e"],
"ExclusiveGateways": ["g0","g1"],
"InclusiveGateways": [],
"ParallelGateways": [],
"SequenceFlows": [["s","A"],["A","g0"],["g0","B"],["g0","C"],["B","g1"],["C","g1"],["g1","e"]]
}
```

Abbildung 7.1: Beispiel für JSON-Syntax für BPMN-Modelle

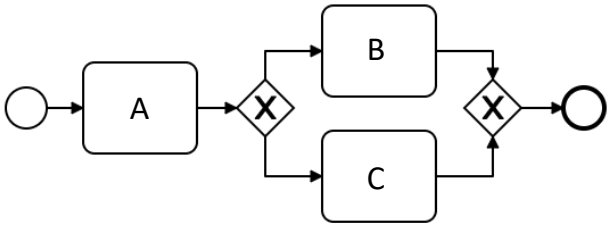


Abbildung 7.2: BPMN-Beispielmodell [11]

```
<GatewayListe> ::= <Gateway> | <Gateway> "," <Gatewayliste>
<SequenzflussListe> ::= <Sequenzfluss> | <Sequenzfluss> "," <Sequenzflussliste>
<Sequenzfluss> ::= "[" <Knoten> "," <Knoten> "]"
<Knoten> ::= <AktivitätsID> | <Ereignis> | <Gateway>
```

Die definierte Syntax für BPMN-Modelle unterstützt alle BPMN-Konstrukte, die für die in dieser Arbeit entwickelten Methoden notwendig sind (Kapitel 3). Zusätzlich wird die Verwendung von inklusiven Gateways ermöglicht. Modelle mit derartigen Gateways werden jedoch nicht unterstützt und sollten immer durch ein leeres Array definiert werden.

Abbildung 7.1 zeigt abschließend ein Beispiel für eine JSON-Datei, die ein BPMN-Modell repräsentiert. Das korrespondierende BPMN-Modell ist in Abbildung 7.2 dargestellt.

7.1.2 Syntax für Declare-Modelle

Für die Eingabe von Declare-Modellen werden Textdateien akzeptiert. Dabei stehen dort zuerst die Namen der im Prozess vorkommenden Aktivitäten jeweils mit einem Komma separiert. Das Ende der Aufzählung der Aktivitäten wird durch ein Semikolon gekennzeichnet. Anschließend folgen die im Prozess vorkommenden Constraints mit belegten Parametern, die jeweils durch ein Semikolon getrennt werden. Die Parameter müssen dabei in der Menge der Aktivitäten definiert sein. Umgekehrt ist es aber natürlich durchaus möglich, dass Aktivitäten definiert werden, die in keinem Constraint vorkommen. In der vorliegenden Implementierung werden sämtliche gängigen

Declare-Constraints [9] unterstützt, wie sie in dieser Arbeit verwendet werden (siehe Kapitel 2.1.4). Für die korrekte Schreibweise der jeweiligen Constraint-Namen wird auf Kapitel 7.2 verwiesen. Hier folgt noch die formale Syntaxdefinition in BNF:

```
<Declare-Modell> ::= <Aktivitäten> ";" <Constraints>
<Aktivitäten> ::= <Aktivität> { "," <Aktivität> }
<Constraints> ::= <Constraint> { ";" <Constraint> }
<Constraint> ::= <Unäres-Constraint> | <Binäres-Constraint>
<Unäres-Constraint> ::= <Constraint-Name> "(" <Aktivität> "," <Ganze-Zahl> ")"
<Binäres-Constraint> ::= <Constraint-Name> "(" <Aktivität> "," <Aktivität> ")"
<Constraint-Name> ::= <beliebiger Constraint-Name>
<Aktivität> ::= <Zeichen>
<Zeichen> ::= <beliebiges Zeichen">
<Ganze-Zahl> ::= <Ziffer> { <Ziffer> }
<Ziffer> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

Ein Beispiel für eine gültige Syntax eines Declare-Modells wird in Abbildung 7.3 gezeigt. Das entsprechende Modell besteht aus den fünf Aktivitäten A, B, C, D, E und fünf verschiedenen Constraints: chainResponse(A, C), succession(A, B), response(D, C), precedence(C, B) sowie notCoExistence(A, D).

```
A, B, C, D, E;
chain_response(A,C);
succession(A,B);
response(D,C);
precedence(C,B);
not_co_existence(A,D);
```

Abbildung 7.3: Beispiel für Syntax für Declare-Modelle

7.1.3 Syntax für deterministische endliche Automaten

Deterministische endliche Automaten werden wie BPMN-Modelle als JSON-Dateien eingelesen und ausgegeben. Die Syntax dieser Dateien folgt dabei streng der mathematischen Definition von DEAs (Kapitel 2.2). Dabei ist zu beachten, dass die Transitionsfunktion als Menge von Tripeln definiert wird. Ein Tripel besteht aus dem Ausgangszustand, dem gelesenen Symbol und dem Zielzustand der Transition. Die formale Syntax in BNF sieht folgendermaßen aus:

```
<DEA> ::= "States:[" <Zustandsmenge> "],"
          "Alphabet:[" <Alphabetmenge> "],"
```

```
            "TransitionFunction:[" <Übergangsfunktionen> "],"
            "StartState:" <Zustand> ","
            "AcceptingStates:" <Zustandsmenge>
<Zustandsmenge> ::= <Zustand> | <Zustand> "," <Zustandsmenge>
<Alphabetmenge> ::= <Zeichen> | <Zeichen> "," <Alphabetmenge>
<Übergangsfunktionen> ::= "[" <Übergang> "]"
                        | "[" <Übergang> "]" "," <Übergangsfunktionen>
<Übergang> ::= <Zustand> "," <Zeichen> "," <Zustand>
<Zustand> ::= Name eines Zustands
<Zeichen> ::= Beliebiges Zeichen
```

Hierzu sei angemerkt, dass sowohl Symbole des Alphabets als auch Namen der Zustände stets in Anführungszeichen zu setzen sind. Diese Information wird aus Übersichtsgründen in der BNF weggelassen. Zudem sei in diesem Zusammenhang darauf hingewiesen, dass die eingegebenen DEAs einer Validierung unterzogen werden, um beispielsweise Nichtdeterminismus auszuschließen. Ungültige Eingaben führen zum Auslösen eines Fehlers. In Abbildung 7.4 wird eine gültige JSON-Eingabedatei für den in Abbildung 7.5 dargestellten DEA illustriert.

```
{
"States": ["q0","q1","q2"],
"Alphabet": ["A","B","C"],
"TransitionFunction": [["q0","A","q1"],["q0","B","q0"],["q0","C","q0"],["q1","A","q2"],
["q1","B","q0"],["q1","C","q2"],["q2","A","q2"],["q2","B","q2"],["q2","C","q2"]],
"StartState": "q0",
"AcceptingStates": ["q0"]
}
```

Abbildung 7.4: Beispiel für Syntax für DEAs

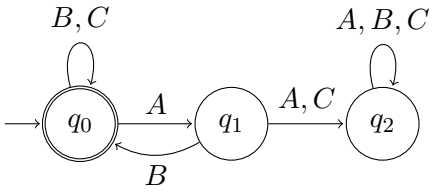


Abbildung 7.5: Beispiel-DEA

7.2 Konsolenanwendung zum Vergleich von Prozessmodellen

In Form einer Konsolenanwendung wurde ein Applikation entwickelt, das die in Kapitel 3.2 beschriebenen Techniken programmatisch umsetzt. Mit deren Hilfe ist es möglich, sowohl Declare-

compare	Überprüft zwei Prozessmodelle (oder DEAs) auf Äquivalenz und gegenseitige Teilmengeneigenschaften.
generate-dfa	Generiert einen DEA für ein gegebenes Modell.
enumerate-accepted-words	Gibt für ein Intervall $[i, j]$ und ein gegebenes Modell (oder DEA) alle validen Sequenzen aus, deren Länge im Intervall liegt.
generate-difference-automata	Gibt für zwei gegebene Modelle (oder DEAs) die Automaten der Differenzen, der symmetrischen Differenz und den Produktautomaten aus.
generate-regex	Generiert einen regulären Ausdruck für ein gegebenes Modell oder Automaten.

Tabelle 7.1: Modi der Konsolenanwendung

Prozessmodelle als auch BPMN-Prozessmodelle

- mit Hilfe des inklusions-basierten Modellvergleichs zu vergleichen (Kapitel 3.2.6),
- auf Teilmengenbeziehungen zu überprüfen (Kapitel 3.2.7),
- im Hinblick auf Gemeinsamkeiten zu untersuchen (Kapitel 3.2.8) und
- in Bezug auf Unterschiede zu analysieren (Kapitel 3.2.9).

Die Applikation wird durch Ausführen einer JAR-Datei über die Konsole aufgerufen. Mit Ausnahme einiger Sonderfälle wie etwa der Hilfeseite besteht jeder Aufruf aus einem Modus, einer variablen Anzahl von Eingabedateien und wahlweisen Optionen. Die Hauptfunktion der Applikation besteht darin, Modelle (und DEAs) bzw. deren Sprachen auf Äquivalenz und Teilmengeneigenschaft zu überprüfen. Darüber hinaus bietet die Applikation mehrere weitere Berechnungsfunktionen und Hilfsfunktionen. Funktionen können auch in einem einzigen Aufruf durch Angabe entsprechender Optionen kombiniert werden. In Tabelle 7.1 sind alle verfügbaren Modi aufgelistet.

Außerdem verfügt die Applikation über einige nützliche Hilfsaufrufe, die ihre Verwendung erleichtern sollen. Beispielsweise gibt der Aufruf **list-constraints** alle in der Implementierung integrierten Declare-Constraints inklusive deren korrekter Syntax aus. Alle Hilfsaufrufe sind in Tabelle 7.2 zu finden. Zudem besteht die Möglichkeit, berechnete Ergebnisse direkt auf der Konsole auszugeben. Die zugehörigen Aufrufe sind in Tabelle 7.3 dargestellt.

-help	Zeigt Hilfeseite.
list-constraints	Zeigt alle unterstützten Declare-Constraints.
show-input-examples	Gibt Beispiele valider Input-Syntax.

Tabelle 7.2: Hilfsaufrufe

-file-output	Generiert Dateien für alle Output-Objekte.
-dfa-output	Gibt die generierten Automaten der Input Modelle auf der Konsole aus.

Tabelle 7.3: Output-Aufrufe

Ein Beispiel für einen Aufruf der Applikation ist in Abbildung 7.6 zu sehen. Dabei werden zwei Modelle verglichen und alle akzeptierten Sequenzen bis zur Länge 2 ausgegeben. Die Berechnung liefert als Ergebnis, dass die beiden zu untersuchenden Modelle nicht gleich sind (*The two DFAs are not equivalent.*). Zudem ist das erste Modell nicht im zweiten Modell enthalten (*L(dfa1) is not a subset of L(dfa2).*). Allerdings ist das zweite Modell eine Teilmenge des ersten (*L(dfa2) is a subset of L(dfa1).*).

Die präsentierte Implementierung umfasst sämtliche erforderlichen Methoden für den Vergleich des Kontrollflusses, die in Kapitel 3.2 eingeführt werden. Dabei werden alle relevanten Aspekte abgedeckt. Es ist möglich, imperative und deklarative Modelle zu vergleichen, auf Teilmengeneigenschaften zu überprüfen, Gemeinsamkeiten zu berechnen und Unterschiede zu bestimmen.

```
> java -jar .\compare_models.jar compare --file-output examples\ab.txt examples\
ab.json --accepted-words-output 0 2
Accepted Words of DFA 0:
0: ""
1: "a", "c"
2: "aa", "cc", "ab", "ac", "ca"


Accepted Words of DFA 1:
0:
1:
2:



The two DFAs are not equivalent.
L(dfa1) is not a subset of L(dfa2).
L(dfa2) is a subset of L(dfa1).
```

Abbildung 7.6: Beispielaufruf für einen Modellvergleich

7.3 Applikation zur szenario-basierten Modellprüfung

Für die szenario-basierte Modellprüfung wurde mit Hilfe des Angular-Frameworks[15] eine Webbrowseranwendung in Java implementiert. Diese unterstützt die drei in Kapitel 6 beschriebenen Verhaltensanalysen:

- Überprüfung der Gültigkeit von Sequenzen (Verhaltensanalyse 1)
- Vervollständigung von Sequenzen (Verhaltensanalyse 2)
- Vervollständigung von Sequenzen innerhalb von n Schritten (Verhaltensanalyse 3)

Abbildung 7.7 zeigt die Startseite der Anwendung. Auf Grund einer noch zu erläuternden, durchgeführten Benutzerstudie (Kapitel 8.6) ist das Werkzeug in deutscher Sprache gehalten. Des Weiteren sind Fachtermini wie „Modell“ oder „Sequenz“ durch die im allgemeinen Sprachgebrauch gängigeren Begriffe „Menge an Regeln“ oder „Zeichenkette“ ersetzt. Die Erklärungen aller Begriffsersetzungen sowie die Gründe hierfür werden in Kapitel 8.6 beschrieben.

Im Folgenden werden der Aufbau des Werkzeugs und die graphische Umsetzung der drei implementierten Verhaltensanalysen erläutert. Zunächst kann mit Hilfe des Buttons „Datei auswählen“ im Feld „Regeln laden“ ein Prozessmodell geladen werden. Hierbei werden entweder Declare-Prozessmodelle oder endliche deterministische Automaten jeweils der Form wie in Kapitel 7.2 beschrieben akzeptiert. Anschließend wird die geladene Datei im Fenster „Menge an Regeln“, das sich rechts daneben befindet, angezeigt. In Abbildung 7.7 ist beispielhaft das erste Modell aus der Benutzerstudie aus Kapitel 8.6 geladen. Dieses besteht aus den fünf Aktivitäten A, B, C, D und E sowie den fünf angezeigten Declare-Constraints.

Das geladene Modell bildet die Grundlage für weitere Untersuchungen in den drei darunter abgebildeten Fenstern. Diese repräsentieren die in Kapitel 6 beschriebenen Verhaltensanalysen und werden nun anhand einiger Aufgaben aus der durchgeführten Benutzerstudie (Kapitel 8.6) erläutert.

In Aufgabe 1 (*„Welche Regeln erfüllt die Zeichenkette „CBEAC“?“*) der Benutzerstudie (Kapitel 8.6) wird Verhaltensanalyse 1 benötigt. Diese ist im ersten der drei Fenster („Zeichenkette auf Gültigkeit prüfen“) umgesetzt. Zum Lösen der Aufgabe gibt man die gewünschte Sequenz in das leere Feld ein und erhält durch einen Klick auf den Button „Gültigkeit prüfen“ das Ergebnis aus Abbildung 7.8 angezeigt. Dort ist zum einen zu sehen, dass die Sequenz (in diesem Fall) ungültig ist. Zusätzlich wird für jede Regel angezeigt, ob sie verletzt oder erfüllt ist. In diesem Fall ist lediglich das succession(A, B) Constraint verletzt. Alle vier weiteren Constraints sind erfüllt.

[15] `https://angular.de`, abgerufen am: 27. Dezember 2023

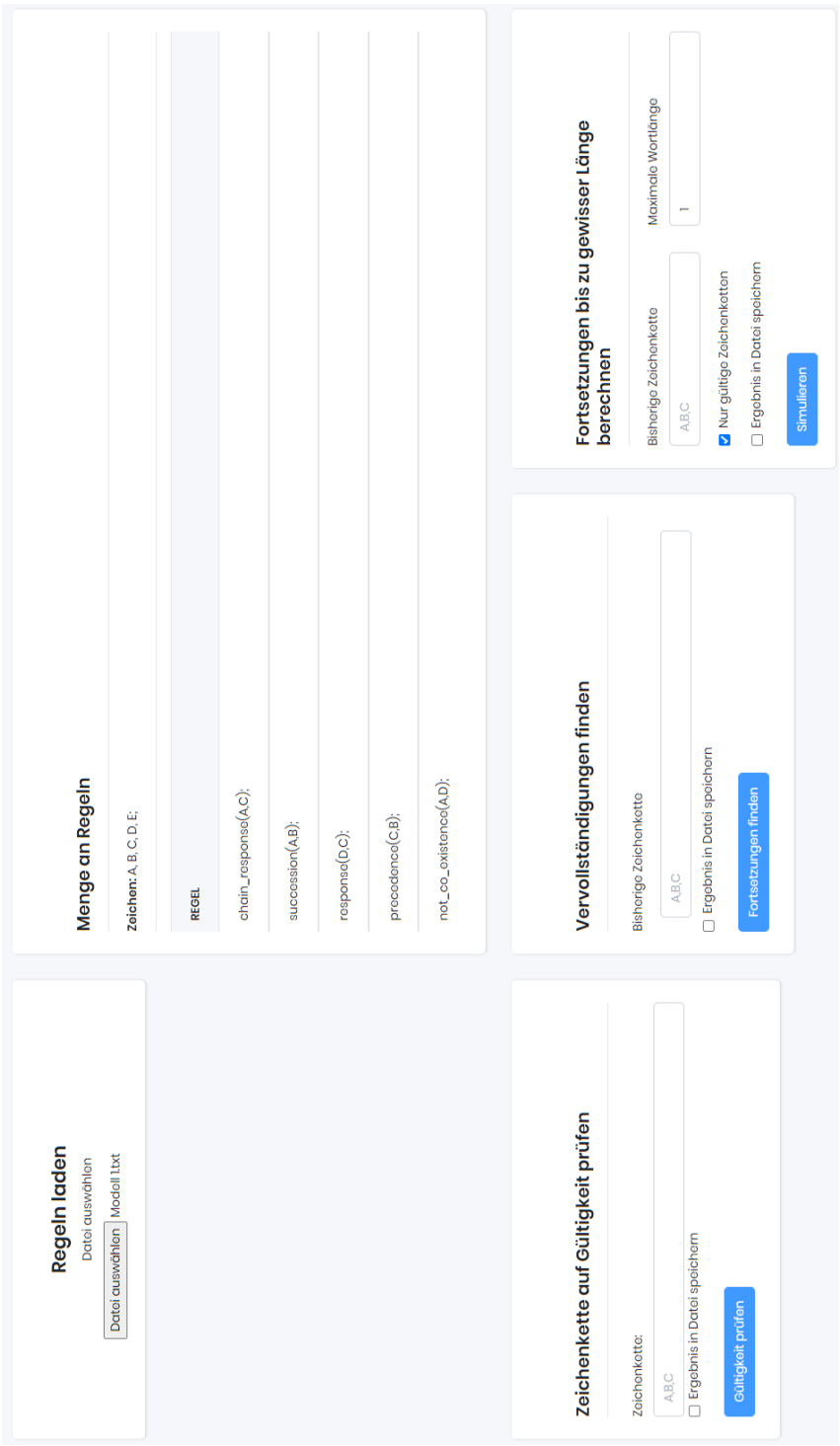


Abbildung 7.7: Startseite des entwickelten Werkzeugs

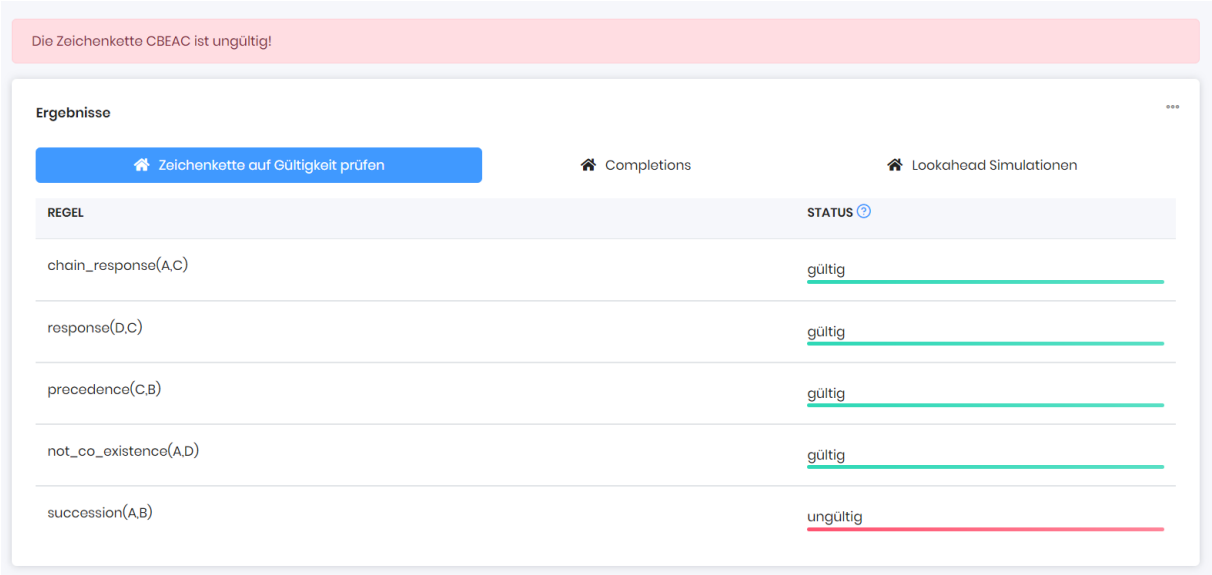


Abbildung 7.8: Überprüfen einer Sequenz auf Gültigkeit (Verhaltensanalyse 1)

Für die Bearbeitung von Aufgabe 2*a* (*„ACC“ ist keine gültige Zeichenkette. Wie kann „ACC“ fortgesetzt werden, damit eine minimale gültige Zeichenkette entsteht?*“) braucht man Verhaltensanalyse 2. Dazu muss im Feld „Bisherige Zeichenkette“ die bisherige Teilsequenz eingetragen werden. Nach Klicken auf den Button „Fortsetzungen finden“ erhält man das Ergebnis aus Abbildung 7.9. In diesem Fall wird „*ACCB*“ als (einzige) minimale Fortsetzung der Sequenz „*ACCB*“ gefunden und zurückgegeben.

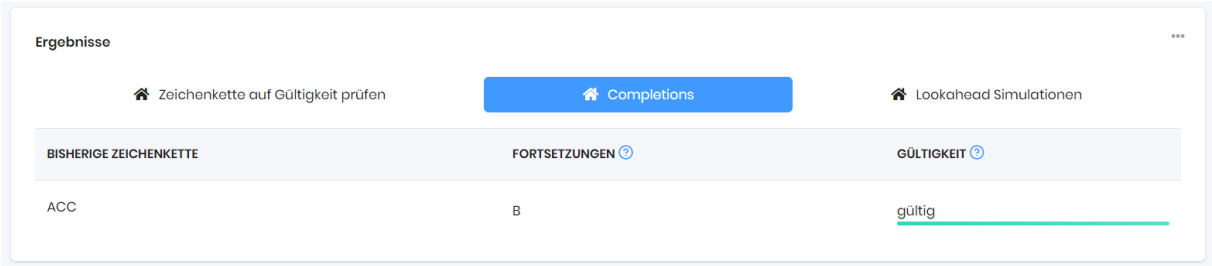


Abbildung 7.9: Fortsetzen einer Teilsequenz (Verhaltensanalyse 2)

Verhaltensanalyse 3 wird durch das dritte Fenster „Fortsetzungen bis zu gewisser Länge berechnen“ unterstützt. Aufgabe 3*a* (*„Geben Sie alle gültigen Zeichenketten bis einschließlich Länge 2 an.*“) erfordert die leere Eingabe im Feld „Bisherige Zeichenkette“ sowie Länge 2 im Feld „maximale Wortlänge“. Durch einen Klick auf den Button „Simulieren“ bekommt man die Anzahl aller gültigen Sequenzen bis einschließlich Länge 2. Diese werden in der Ausgabe einzeln aufgelistet (Abbildung 7.10). Die weiteren drei nicht abgebildeten Sequenzen erhält man durch

Scrollen in der Applikation.

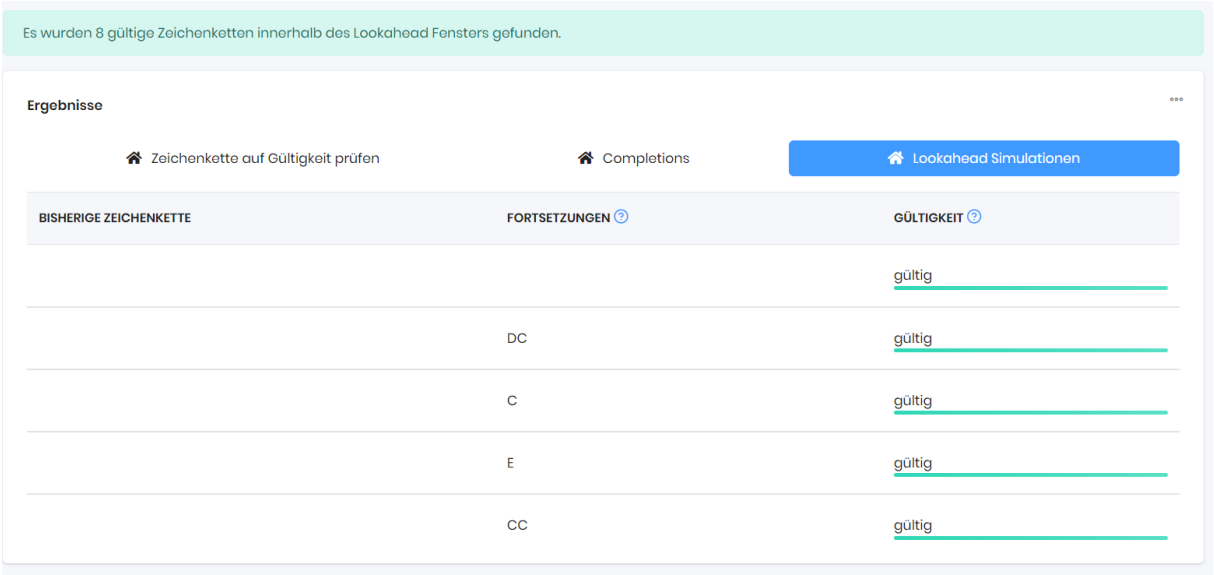


Abbildung 7.10: Alle gültigen Fortsetzungen bis zu gewisser Länge (Verhaltensanalyse 3)

Für alle drei Verhaltensanalysen kann man optional noch einen Haken „Ergebnis in Datei speichern“ setzen. Falls dieser Haken gesetzt ist, wird das Ergebnis in einer Textdatei gespeichert und im Ordner, in dem die ausgeführte JAR-Datei liegt, abgelegt.

Die vorliegende Implementierung realisiert erfolgreich alle drei Verhaltensanalysen und präsentiert die Ergebnisse auf anschauliche Weise durch grafische Darstellungen. Dies ermöglicht eine umfassende und leicht verständliche Visualisierung der durchgeführten Verhaltensanalysen.

Kapitel 8

Evaluation

In diesem Abschnitt erfolgt eine detaillierte Bewertung der in den Kapiteln 3 bis 6 entwickelten Konzepte und Methoden, wobei der Schwerpunkt auf der Unterscheidung zwischen theoretischen Überlegungen und empirischen Erkenntnissen liegt. Dabei werden verschiedene Aspekte berücksichtigt, darunter die praktische Anwendbarkeit in realen Szenarien, die theoretische Laufzeit und die Nützlichkeit für die Endnutzerin oder den Endnutzer.

Zuerst werden die beiden Methoden zum Vergleich von Prozessmodellen (Kapitel 3.2.4 und 3.2.6) in Kapitel 8.1 einzeln evaluiert und gegenübergestellt. Der Fokus liegt hier auf der theoretischen Analyse ihrer Funktionsweise und Effektivität.

Anschließend erfolgt die detaillierte Bewertung des Konzepts zur Berechnung von Verhaltensänderungen in deklarativen Prozessmodellen (Kapitel 4). Die Grundlage dieser Bewertung bildet eine gründliche Analyse mit Hilfe mehrerer realer Datensätze (Kapitel 8.2). Hierbei wird der Fokus auf die praktische Anwendbarkeit und den Beitrag zur Theorie gelegt.

Des Weiteren bietet Abschnitt 8.3 Interpretationen der in Kapitel 5 definierten Ähnlichkeitsmaße sowohl für Beispielprozesse als auch für Prozessmodelle, die aus realen Datensätzen extrahiert werden. Hierbei liegt der Schwerpunkt auf der theoretischen Fundierung und der praktischen Relevanz.

Schließlich werden die Verhaltensanalysen, die in Kapitel 6 definiert und beschrieben sind, genauer untersucht. In Kapitel 8.4 werden zunächst die theoretischen Laufzeiten aller drei Verhaltensanalysen bestimmt, um einen fundierten Einblick in ihre algorithmischen Komplexitäten zu gewinnen. Diese theoretischen Überlegungen dienen als Grundlage, um potenzielle Engpässe oder Herausforderungen in der Laufzeitperformance frühzeitig zu identifizieren und ermöglichen somit eine gezielte Optimierung der Methoden für ihre effiziente Anwendung in realen Szenarien.

Eine anschließende empirische Evaluation der Verhaltensanalysen anhand realer Datensätze in Abschnitt 8.5 zeigt deren praktische Anwendbarkeit. Hier wird die Balance zwischen theoretischen

Überlegungen und empirischen Ergebnissen herausgestellt.

Abschließend verdeutlicht eine durchgeführte Benutzerstudie in Kapitel 8.6 die Vorteile der automatisierten Unterstützung durch die Verhaltensanalysen für die Endnutzerin oder den Endnutzer. Hierbei liegt der Fokus auf der praktischen Anwendbarkeit und den konkreten Vorteilen, die Anwenderinnen und Anwender aus den entwickelten Methoden ziehen können.

Die durchgeführten Evaluierungen verdeutlichen sowohl die theoretische Fundierung als auch die praktische Anwendbarkeit der entwickelten Ansätze und Methoden. Die Laufzeiten der Methoden erweisen sich als äußerst praxistauglich für reale Anwendungen, wobei die Effizienz und Geschwindigkeit einen klaren Mehrwert bieten. Darüber hinaus zeigt sich der Nutzen der entwickelten Ansätze besonders deutlich für die Endnutzerin oder den Endbenutzer, da sie essentielle Verbesserungen und Vorteile in der Anwendung von Prozessmodellen ermöglichen. Diese Ergebnisse unterstreichen die Relevanz und Effektivität der entwickelten Methoden im Hinblick auf ihre direkte Anwendbarkeit und den Mehrwert, den sie bei der Anwendung in praktischen Szenarien bieten können.

8.1 Simulationsbasierter Ansatz vs. inklusionsbasierter Ansatz

Die in Kapitel 3 entwickelten Verfahren des simulationsbasierten (Kapitel 3.2.4) und des inklusionsbasierten Ansatzes (Kapitel 3.2.6) liefern prinzipiell ein und dasselbe Resultat: Beide entscheiden, ob zwei Prozessmodelle die gleichen Sequenzen akzeptieren und entscheiden somit über die Gleichheit der beiden Modelle. Allerdings sind die jeweiligen Vorgehensweisen komplett verschieden und gewähren folglich differenzierte Einblicke in die zu Grunde liegenden Prozessmodelle. Der wesentliche Unterschied zwischen den beiden Algorithmen besteht darin, dass beim simulationsbasierten Ansatz konkrete Sequenzen bis zu einer gewissen Länge (*upper bound*, Kapitel 3.2.4) berechnet werden, wohingegen beim inklusionsbasierten Ansatz über die Gleichheit der Modelle mit Hilfe von abstrakten Automaten entschieden wird. Dies hat zur Folge, dass sich die beiden Algorithmen gravierend in ihren Laufzeiten unterscheiden. In diesem Kapitel werden alle Unterschiede zwischen den Algorithmen hinsichtlich ihrer Laufzeit und potenziellen Anwendungsfälle detailliert herausgearbeitet und umfassend erläutert. Zudem erfolgt eine Empfehlung, in welchen Szenarien der jeweilige Vergleichsalgorithmus bevorzugt eingesetzt werden sollte.

Zunächst werden die Laufzeiten der Algorithmen *Simulation-based algorithm* (Algorithmus 2) und *Inclusion-based algorithm* (Algorithmus 3) miteinander verglichen. Beide Algorithmen beginnen gleich: Zunächst werden für beide Prozessmodelle die repräsentierenden Prozessautomaten berechnet. Daher ist für den Unterschied zwischen den beiden Laufzeiten nur der eigentliche Gleichheitsvergleich von Relevanz.

Beim simulationsbasierten Ansatz wird zunächst die *upper bound* b berechnet: Dafür muss lediglich das Produkt $b := |Q_1| \cdot |Q_2|$ der Anzahlen der Zustände der beiden Prozessautomaten $M_1 = (\mathcal{A}, Q_1, s_{0_1}, \delta_1, F_1)$ und $M_2 = (\mathcal{A}, Q_2, s_{0_2}, \delta_2, F_2)$ bestimmt werden [21]. Die Berechnung dieses einfachen Produkts ist laufzeittechnisch zu vernachlässigen.

Als Nächstes folgt der laufzeittechnisch aufwendigste Part des simulationsbasierten Ansatzes: Das Generieren aller Sequenzen bis zur Länge b, einschließlich des anschließenden Testens auf Akzeptanz bezüglich der beiden Prozessmodelle, erfordert exponentielle Laufzeit. Für eine Sequenz der Länge n über einer endlichen Menge von Aktivitäten $\mathcal{A}$ gibt es $|\mathcal{A}|^n$ verschiedene Möglichkeiten. Demzufolge erhält man

$$\sum_{i=0}^{b} |\mathcal{A}|^i$$

Sequenzen der Länge kleiner gleich der *upper bound* b [21]. Offensichtlich steigt dieser Wert exponentiell in der Länge der *upper bound* an. Analog müssen genauso viele Sequenzen auf Akzeptanz überprüft werden, was wiederum eine exponentielle Laufzeit zur Folge hat. Eine Alternativlösung wäre, wie in Kapitel 3.2.5 angesprochen, einen SAT-Solver [98] (z.B. Alloy[16]) zu verwenden. Allerdings können diese Ansätze ebenso exponentielle Laufzeit aufweisen, da SAT-Solving für Aussagenlogik nach dem Cook-Levin Theorem [99] NP-vollständig ist [100].

Im Gegensatz zu simulationsbasierten Ansatz ist es beim inklusionsbasierten Ansatz nicht notwendig, Sequenzen zu simulieren. Hierbei wird lediglich der Automat für das symmetrische Produkt $M_1 \Delta M_2$ der beiden Prozessautomaten $M_1 = (\mathcal{A}, Q_1, s_{0_1}, \delta_1, F_1)$ und $M_2 = (\mathcal{A}, Q_2, s_{0_2}, \delta_2, F_2)$ berechnet und anschließend minimiert. Die Berechnung des Produktautomaten verläuft dabei in polynomieller Laufzeit $\mathcal{O}(|Q_1| \cdot |Q_2|)$ [50]. Die anschließende Minimalisierung des Produktautomaten durch den Hopcroft-Algorithmus erfolgt in logarithmischer Laufzeit $\mathcal{O}(|Q_1| \cdot |Q_2| \cdot \log(|Q_1| \cdot |Q_2|))$ [51]. Daraus ergibt sich für den inklusionsbasierten Vergleichsalgorithmus (ohne Konstruktion der Prozessautomaten) eine asymptotische Gesamtlaufzeit von

$$\mathcal{O}(|Q_1| \cdot |Q_2|) + \mathcal{O}(|Q_1| \cdot |Q_2| \cdot \log(|Q_1| \cdot |Q_2|)) = \mathcal{O}(|Q_1| \cdot |Q_2|) \text{ [22].}$$

Insgesamt gesehen ist der inklusionsbasierte Algorithmus in Bezug auf die Laufzeiteffizienz dem simulationsbasierten Algorithmus deutlich überlegen. Daher ist der erstgenannte Ansatz zweifellos die bessere Wahl, wenn es darum geht, zwei gegebene Prozessmodelle auf Gleichheit zu überprüfen [22]. Dennoch kann die Anwendung des simulationsbasierten Ansatzes bei ungleichen Prozessmodellen wertvolle Einblicke bieten. Oftmals ist es nicht erforderlich, alle Sequenzen bis zur *upper bound* zu simulieren, da signifikante Unterschiede bereits bei geringen Sequenzlängen auftreten. Darüber hinaus ermöglicht der simulationsbasierte Ansatz eine eingehende Analyse

[16] `http://www.alloytools.org`, abgerufen am: 27. Dezember 2023

des Verhaltens der Prozessmodelle, indem konkrete akzeptierte und nicht-akzeptierte Sequenzen betrachtet werden. Im Gegensatz dazu liefert der inklusionsbasierte Ansatz lediglich einen abstrakten Automaten [22], der schwer zu interpretieren ist.

Basierend auf den oben genannten Beobachtungen wird folgende Vorgehensweise empfohlen [22]: Zuerst sollte der effiziente inklusionsbasierte Algorithmus angewandt werden, um eine Entscheidung über die Gleichheit der Prozessmodelle zu treffen. Abhängig vom spezifischen Anwendungsfall und den Intentionen der Benutzerin bzw. des Benutzers kann anschließend – jedoch natürlich nur im Fall, dass die Prozessmodelle als ungleich erkannt werden – der simulationsbasierte Ansatz verwendet werden. Auf diese Weise kann nicht nur festgestellt werden, dass die Modelle unterschiedlich sind, sondern es werden auch konkrete Ergebnisse in der Form von akzeptierten und nicht-akzeptierten Sequenzen erzeugt. Diese Ergebnisse veranschaulichen das Resultat und bieten somit Einblicke in die Verhaltensweisen und damit in die Unterschiede der beiden Prozessmodelle [22].

Bei diesem Vorgehen handelt es sich lediglich um eine Empfehlung. Letztendlich obliegt es der Benutzerin oder dem Benutzer selbst, zu entscheiden, in welchem Maße die bereitgestellten Verfahren genutzt werden. Die eigene Erfahrung zeigt allerdings, dass sich beide Ansätze gegenseitig gut ergänzen, um zum Teil komplexe Prozessmodelle zu vergleichen und zu verstehen [22].

8.2 Evaluation der Berechnung von Verhaltensänderungen

In diesem Abschnitt erfolgt die Evaluierung der in Kapitel 4 ausgearbeiteten Methode zur Berechnung von Verhaltensänderungen in deklarativen Prozessmodellen. Diese Evaluierung wird mithilfe der in Abschnitt 7.2 beschriebenen Implementierung auf realen Datensätzen durchgeführt. Dazu wurden die letzten vier Datensätze der *Business Process Intelligence Challenge (BPIC)*[17], nämlich die Event Logs *BPIC 2017 (financial industry), BPIC 2018 (fund process), BPIC 2019 (purchase process)* und *BPIC 2020 (reimbursement process)* verwendet.[18] Der *BPIC 2020* Event Log besteht dabei aus fünf Sublogs, nämlich *BPIC'20(domestic)*, *BPIC'20(international)*, *BPIC'20(permits)*, *BPIC'20(requests)* und *BPIC'20(prepaid)*. Aus allen Datensätzen wurden mit Hilfe des MINERful Miners [101, 102] Declare-Modelle extrahiert. Als Mining-Parameter wurden dabei die in [15] empfohlenen Werte verwendet: Sowohl *support* (Kapitel 8.3.2), *confidence* (Kapitel 8.3.2) als auch *interest* [101, 102] wurden auf 75% gesetzt. Eine Ausnahme wurde für den *BPIC 2017 (financial industry)* Event Log gemacht. Da bei diesem die Parameter $75\%, 75\%, 75\%$ zu unpraktikablen Prozessmodellen führten, die aus mehreren hunderten Constraints bestehen,

[17] `https://www.tf-pm.org/competitions-awards/bpi-challenge`, abgerufen am: 27. Dezember 2023
[18] `https://data.4tu.nl`, abgerufen am: 27. Dezember 2023

Datensatz	#Constraints
BPIC 2017	154
BPIC 2018	178
BPIC 2019	14
BPIC'20(domestic)	32
BPIC'20(int.)	116
BPIC'20(permits)	70
BPIC'20(requests)	32

Tabelle 8.1: Übersicht über extrahierte Prozessmodelle [23]

wurden für diesen Event Log die Werte *support*, *confidence* sowie *interest* auf $95\%, 95\%, 95\%$ gesetzt [23]. Alle extrahierten Prozessmodelle sind online zu finden.[19]

Beim manuellen Überprüfen der extrahierten Prozessmodelle wurde festgestellt, dass das zu *BPIC'20(prepaid)* gehörende Prozessmodell nur eine einzige Sequenz akzeptiert. Diese Beobachtung lässt die Vermutung zu, dass das Prozessmodell überspezifiziert und daher für weitere Berechnungen uninteressant ist. Aus diesem Grund wurde dieser Event Log aus der Evaluation ausgeschlossen. Tabelle 8.1 zeigt die Größen des zum jeweiligen Event Logs extrahierten Declare-Modells [23]. Auf diesen extrahierten Prozessmodellen wurde dann die Methode zur Berechnung von Verhaltensänderungen evaluiert. Dafür wurden aus jedem Prozessmodell per Zufall 25% aller Constraints gelöscht. Dies soll eine Veränderung des Modells um 25% simulieren. Da das Hinzufügen von Constraints lediglich zur Vertauschung der Rollen in beiden Modellen führen würde (das Löschen eines Constraints in einem Modell entspricht dem Hinzufügen desselben Constraints im anderen Modell) und das Ersetzen von Constraints sowohl eine Lösch- als auch eine Hinzufügeoperation repräsentiert, ist dieses Vorgehen ausreichend, um alle möglichen Änderungen abzudecken [23]. Daraus kann geschlossen werden, dass durch die Änderungen nur neue Sequenzen hinzukommen und keine Sequenzen verloren gehen können. Somit muss in diesem Fall jeweils nur ein Differenzautomat, nämlich derjenige, der das neu hinzugekommene Verhalten beschreibt, berechnet werden. Alle Differenzautomaten sind online auf GitHub[20]zu finden. Schließlich wurden alle Sequenzen bis zu einer Sequenzlänge von 10, die der Differenzautomat akzeptiert, berechnet. Alle Berechnungen wurden auf einem Mac OS-System mit einem 3 GHz-Prozessor und 16 GB RAM durchgeführt.

Die für die Berechnung der Sequenzen (Verhaltensänderungen) benötigten Laufzeiten sowie die Anzahl der berechneten Sequenzen sind in Tabelle 8.2 dargestellt [23]. Wie man erkennen kann, konnten die meisten Berechnungen - selbst im Falle von tausenden von Sequenzen - in weniger

[19] `https://github.com/NicolaiSchuetzenmeier/Computation-Behavioral-Changes`, abgerufen am: 27. Dezember 2023

[20] `https://github.com/NicolaiSchuetzenmeier/Computation-Behavioral-Changes`, abgerufen am: 27. Dezember 2023

Datensatz	Laufzeit	#Sequenzen
BPIC 2017	0.69	2
BPIC 2018	0.76	1
BPIC 2019	1.12	$> 1,000$
BPIC'20(domestic)	0.53	55
BPIC'20(int.)	0.72	26
BPIC'20(permits)	41.21	$> 32,000$
BPIC'20(requests)	0.82	251

Tabelle 8.2: Laufzeiten (in Sekunden) zur Berechnung der Verhaltensänderungen (Löschen von 25% der Constraints) und Anzahl der berechneten Sequenzen bis zur Länge 10 pro Event Log [23]

als einer Sekunde durchgeführt werden. Die längste Berechnung dauerte etwa 41 Sekunden, was sich immer noch in einem durchaus akzeptablen Rahmen bewegt. Dies deutet darauf hin, dass es für alle betrachteten Datensätze möglich ist, Verhaltensänderungen in einer „live“ Umgebung, z.B. während des Re-Modellierens, zu berechnen.

Um ein besseres Verständnis für die Skalierbarkeit des Ansatzes zu erlangen, wurden obige Berechnungen mit Sequenzlängen in Zweierschritten von $2, 4, \ldots, 16$ und verschiedenen Prozentsätzen für die Änderungen der Prozessmodelle durchgeführt. Dabei wurden jeweils per Zufall $10\%, 20\%, \ldots, 50\%$ der Constraints gelöscht. Abbildung 8.1 zeigt beispielhaft die Resultate für den Event Log *BPIC 2019* [23]. Man erkennt, dass die Laufzeit hauptsächlich von der (von der Benutzerin oder dem Benutzer definierten) Sequenzlänge abhängt. Im Gegensatz dazu scheint der tatsächliche Prozentsatz der Löschungen von Constraints keine starke Auswirkung auf die Laufzeit zu haben. Dies sieht man daran, dass es keine große Steigung in Richtung der x-Achse gibt.

Abbildung 8.1 veranschaulicht die Grenzen des Ansatzes, zeigt aber auch seine tatsächlichen Anwendungsgebiete auf [23]: Der Fokus liegt nicht darauf, Verhaltensänderungen (in der Form von konkreten Sequenzen) für sehr große Sequenzlängen zu berechnen. Vielmehr liefert dieser Ansatz bessere Ergebnisse für kleinere Sequenzlängen, z.B. bis zu einer Länge von 10. Diese Einschränkung entspricht auch den tatsächlichen Sequenzlängen in den analysierten Datensätzen, wie beispielsweise im gezeigten *BPIC 2019*-Datensatz, bei dem die durchschnittliche Sequenzlänge bei $6,33$ liegt. Es liegt also nahe, dass die Berücksichtigung von Verhaltensänderungen bis zu einer Sequenzlänge von etwa 10 bereits alle relevanten Informationen über das Prozessmodell liefern kann. Da es jedoch einige (geringe) Auswirkungen auf die Laufzeit im Verhältnis zur Änderungsrate gibt, stellt sich die Frage, ob der Ansatz zum effizienten Vergleich von beliebigen deklarativen Prozessmodellen eingesetzt werden kann. Trotzdem werden damit hervorragende Ergebnisse in konkreten Anwendungsfällen erzielt, bei denen nur ein geringer Prozentsatz der

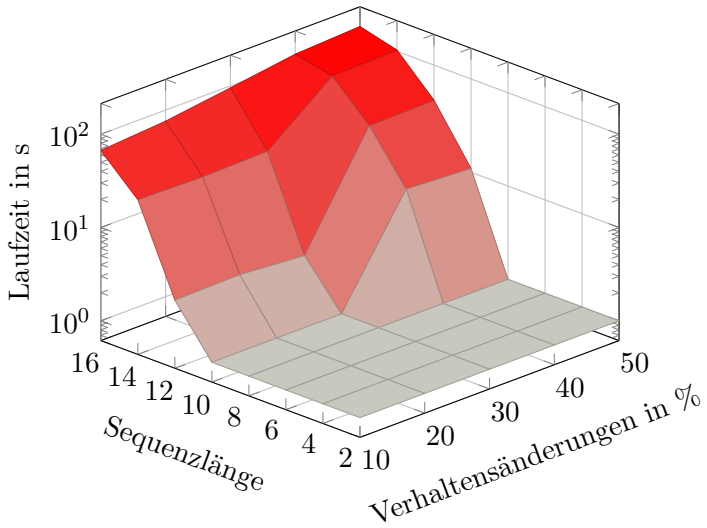


Abbildung 8.1: Laufzeiten (in Sekunden) für Berechnung der Verhaltensänderungen für *BPIC 2019* in Abhängigkeit von der Sequenzlänge und den Verhaltensänderungen in % [23]

Constraints geändert wird.

Tabelle 8.2 enthält die Anzahlen der berechneten Verhaltensänderungen (d.h. Sequenzen) bis zu einer Sequenzlänge von 10 [23]. In einigen Fällen wurden lediglich 1 oder 2 Sequenzen gefunden. In diesen Fällen wäre es durchaus sinnvoll, wenn eine Expertin oder ein Experte diese Sequenzen manuell prüfen würde, um mögliche ungewollte Verhaltensweisen zu identifizieren. Beim *BPIC'20 (permit)* Event Log wurden jedoch über $32,000$ Sequenzen gefunden, was eine reine manuelle Überprüfung schier unmöglich macht. Hier kann die Visualisierungstechnik aus [76] genutzt werden, um eine repräsentative Auswahl dieser $32,000$ Sequenzen zu erhalten. Ein Beispiel hierfür ist in Abbildung 8.2 dargestellt [23], die die repräsentativen Sequenzen für eine Sequenzlänge von 10 zeigt. Eine solche Visualisierung ermöglicht einen systematischen Untersuchungsansatz. Folgende Schritte erleichtern diese Analyse:

1. **Analyse von Start- und Endaktivitäten:** Überprüfen, ob bestimmte Aktivitäten häufig am Anfang oder am Ende auftreten, um potenzielle Start- oder Endaktivitäten zu identifizieren.

2. **Häufigkeit von Aktivitäten:** Untersuchen der Häufigkeit einzelner Aktivitäten über die Zeilen (d.h. Sequenzen), um konsistente „Schlüsselaktivitäten“ zu identifizieren.

3. **Mustererkennung:** Identifizieren von Mustern wie repetitive Sequenzen oder Aktivitätsabfolgen, die auf regelmäßige Prozessschritte oder Schleifen hinweisen können.

4. **Vermutungen überprüfen:** Fokussieren auf spezifische Vermutungen und Prüfung, ob

Abbildung 8.2: Visualisierung der repräsentativen Änderungen für den *BPIC'20 (permit)* Event Log für eine Sequenzlänge von 10 [23]

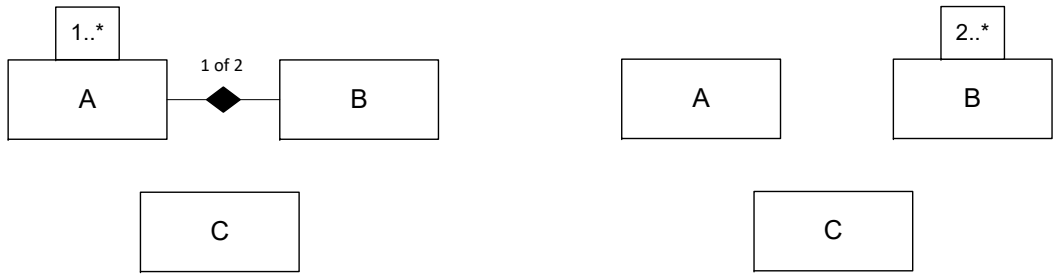
bestimmte Aktivitäten in Verbindung mit anderen auftreten.

5. **Ausreißer erkennen:** Achten auf ungewöhnliche oder seltene Aktivitäten, die auf potenzielle Ausreißer im Prozess hinweisen.

So erkennt man in Abbildung 8.2, dass alle Sequenzen mit der Aktivität *PermitSUBMITTEDby-EMPLOYEE* beginnen (Schritt 1). Eventuell könnte dies ungewolltes Verhalten sein und wäre folglich zu untersuchen. Weiterhin erkennt man in der zweiten Zeile, dass Sequenzen, bei denen die Aktivität *DeclarationFINAL_APPROVED* zweimal am Ende des Prozesses ausgeführt wird, nach der Änderung erlaubt sind (Schritt 3). Dies könnte unter Umständen ungewolltes Verhalten sein und somit eine Überarbeitung des Prozessmodells erfordern.

8.3 Evaluation der Ähnlichkeitsmaße

Die in Abschnitt 5 eingeführten Maße zur Bestimmung von Ähnlichkeiten von Prozessmodellen sollen im Folgenden diskutiert und evaluiert werden. Dabei werden zunächst (Kapitel 8.3.1) an einem übersichtlichen Beispiel die Verwendbarkeit und Ausdruckskraft dieser Maße illustriert. Anschließend (Kapitel 8.3.2) werden die Ähnlichkeitsmaße auf Prozessmodelle, die durch



(a) Prozessmodell Q_1 (b) Prozessmodell Q_2

Abbildung 8.3: Graphische ConDec-Repräsentation der Declare-Prozessmodelle Q_1 und Q_2 [22]

verschiedene Declare Miner generiert wurden, angewandt, um deren Anwendbarkeit auf reale Prozessmodelle zu demonstrieren.

8.3.1 Evaluation der Maße auf Beispielprozessen

In diesem Kapitel werden die Ähnlichkeitsmaße für die bereits in Abschnitt 5.5 kurz eingeführten Declare-Modelle Q_1 und Q_2 berechnet und interpretiert [22]. Diese sind gegeben durch $Q_1 = (\{A, B, C\}, \mathcal{S}_1)$ und $Q_2 = (\{A, B, C\}, \mathcal{S}_2)$, wobei

$$\mathcal{S}_1 = \{\text{existence(A,1), exclusiveChoice(A,B)}\} \text{ und } \mathcal{S}_2 = \{\text{existence(B, 2)}\}.$$

Abbildung 8.3 zeigt zudem die graphische ConDec-Repräsentation von Q_1 und Q_2. Es ist offensichtlich, dass die beiden Modelle verschiedene Prozesse beschreiben, da in Prozessmodell Q_1 Aktivität A mindestens einmal ausgeführt werden muss, wohingegen dies in Prozessmodell Q_2 nicht gefordert wird. Somit akzeptiert Q_2 beispielsweise die Sequenz $\langle BB \rangle$, welche von Q_1 nicht akzeptiert wird, da diese Aktivität A nicht enthält. Folglich ist für alle Ähnlichkeitsmaße ein Wert < 1 zu erwarten.

Tabelle 8.3 zeigt die Werte aller Ähnlichkeitsmaße bis zur Sequenzlänge $n = 7$ [22]. Diese wurden mit Hilfe der entwickelten Implementierung (Kapitel 7.2) berechnet. Zusätzlich sind die zugehörigen Werte der Dichtemaße in Abbildung 8.4 graphisch dargestellt. Daraus lassen sich grundlegende Eigenschaften und Tendenzen der Prozessmodelle ableiten, welche im Folgenden beschrieben werden. Dazu werden zunächst die Werte der einzelnen Maße erläutert und diskutiert. Anschließend wird auf Zusammenhänge zwischen den Ähnlichkeitsmaßen eingegangen, da für eine ausreichende Analyse von Prozessmodellen das Gesamtbild der Maße betrachtet werden muss.

Zunächst werden die n-Dichten $\lambda_n(Q_1)$ und $\lambda_n(Q_2)$ der beiden Modelle Q_1 und Q_2 untersucht. Die leere Sequenz ist die einzige mit der Länge $n = 0$. Diese wird weder von Q_1 noch von Q_2

n	0	1	2	3	4	5	6	7
$\lambda_n(Q_1)$	$\frac{0}{1}$	$\frac{1}{3}$	$\frac{3}{9}$	$\frac{7}{27}$	$\frac{15}{81}$	$\frac{31}{243}$	$\frac{63}{729}$	$\frac{127}{2187}$
$\lambda_n(Q_2)$	$\frac{0}{1}$	$\frac{0}{3}$	$\frac{1}{9}$	$\frac{7}{27}$	$\frac{33}{81}$	$\frac{131}{243}$	$\frac{473}{729}$	$\frac{1611}{2187}$
$\lambda_0^n(Q_1)$	$\frac{0}{1}$	$\frac{1}{4}$	$\frac{4}{13}$	$\frac{11}{40}$	$\frac{26}{121}$	$\frac{57}{364}$	$\frac{120}{1093}$	$\frac{247}{3280}$
$\lambda_0^n(Q_2)$	$\frac{0}{1}$	$\frac{0}{4}$	$\frac{1}{13}$	$\frac{8}{40}$	$\frac{41}{121}$	$\frac{172}{364}$	$\frac{645}{1093}$	$\frac{2256}{3280}$
$\Lambda_n(Q_1, Q_2)$	0	0	0	0	0	0	0	0
$\Lambda_n(Q_2, Q_1)$	0	0	0	0	0	0	0	0
$\Lambda_0^n(Q_1, Q_2)$	0	0	0	0	0	0	0	0
$\Lambda_0^n(Q_2, Q_1)$	0	0	0	0	0	0	0	0
$\Gamma_0^n(Q_1, Q_2)$	-	-	0	0.33	0.5	0.6	0.67	0.71
$\Gamma_0^n(Q_2, Q_1)$	-	-	0	0.29	0.43	0.51	0.56	0.6

Tabelle 8.3: Ähnlichkeitsmaße für Prozessmodelle Q_1 und Q_2 [22]

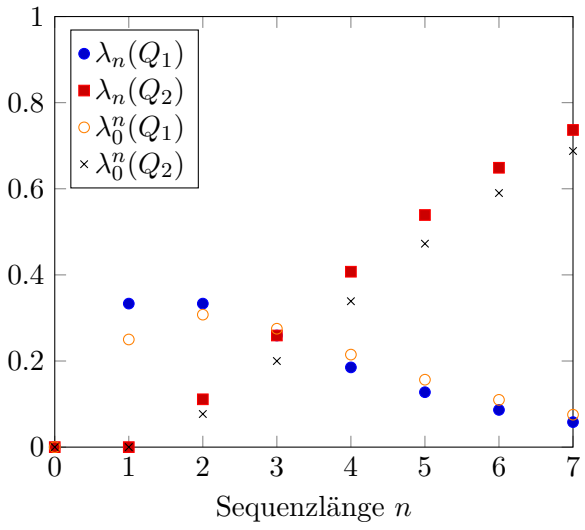


Abbildung 8.4: Dichtemaße für Prozessmodelle Q_1 und Q_2 [22]

akzeptiert, da für Q_1 mindestens einmal die Ausführung der Aktivität A und für Q_2 mindestens zweimal die Ausführung der Aktivität B fehlt. Daraus ergibt sich, dass $\lambda_0(Q_1) = \lambda_0(Q_2) = \frac{0}{1} = 0$.

Alle möglichen Sequenzen der Länge $n = 1$ sind $\langle A\rangle, \langle B\rangle$ und $\langle C\rangle$. Da Q_2 mindestens die zweifache Ausführung von Aktivität B fordert, wird keine Sequenz der Länge $n = 1$ akzeptiert, was eine entsprechende 1-Dichte von $\lambda_1(Q_2) = \frac{0}{3} = 0$ impliziert. Q_1 akzeptiert lediglich Sequenz $\langle A\rangle$ der Länge $n = 1$, da mindestens einmal die Ausführung von Aktivität A gefordert wird. Demnach erhält man eine 1-Dichte von $\lambda_1(Q_1) = \frac{1}{3}$.

Aufgrund der notwendigen zweimaligen Ausführung der Aktivität B akzeptiert Q_2 nur eine Sequenz der Länge $n = 2$, nämlich $\langle BB\rangle$, was eine 2-Dichte von $\lambda_2(Q_2) = \frac{1}{3^2} = \frac{1}{9}$ liefert. Modell Q_1 fordert lediglich die einmalige Ausführung der Aktivität A, wodurch die Ausführung von Aktivität B durch das exclusiveChoice(A, B) Constraint verboten wird. Demzufolge bleiben für Q_1 drei erlaubte Sequenzen der Länge $n = 2$: $\langle AA\rangle, \langle AC\rangle$ und $\langle CA\rangle$. Die entsprechende 2-Dichte liegt daher bei $\lambda_2(Q_1) = \frac{3}{9} = \frac{1}{3}$.

Für $n = 3$ akzeptieren beide Prozessmodelle jeweils sieben verschiedene Sequenzen. Somit gilt für die 3-Dichte, dass $\lambda_3(Q_1) = \frac{7}{27} = \lambda_3(Q_2)$.

Insgesamt gesehen gilt also, dass $\lambda_n(Q_1) \geq \lambda_n(Q_2)$ für $n \leq 3$. Dies liegt daran, dass bei Modell Q_2 durch das existence$(B, 2)$ Constraint bereits zwei Schritte vorgegeben sind, was im Verhältnis zu der kleinen Sequenzlänge (≤ 3) doch erheblich ist. Bei Modell Q_1 ist lediglich ein Schritt durch das existence$(A, 1)$ Constraint vorgegeben. Demzufolge bleiben hier größere Freiheiten und somit mehr mögliche Sequenzen als bei Q_2.

In Tabelle 8.3 erkennt man außerdem, dass $\lambda_n(Q_1) < \lambda_n(Q_2)$ für $n = 4, \ldots, 7$ gilt. Dies kann folgendermaßen interpretiert werden: Prozessmodell Q_1 ist in gewisser Weise „restriktiver“ als Q_2. Das bedeutet, dass Prozessmodell Q_2 während der Ausführung mehr Freiheiten besitzt. Dies lässt sich auch anhand folgender Beobachtung untermauern: Durch das Zusammenspiel des exlusiveChoice(A, B) Constraints und des existence$(A, 1)$ Constraints ist bei Q_1 die Ausführung der Aktivität B verboten. Daher sind in jedem Schritt nur die Aktivitäten A und B erlaubt, wohingegen bei Q_2 prinzipiell alle drei Aktivitäten ausgeführt werden können. Ab $n = 4$ gibt es daher bei Q_2 mehr mögliche Sequenzen als bei Q_1, obwohl bei Q_2 aufgrund des existence$(B, 2)$ bereits zwei Aktivitäten vorgegeben sind (im Gegensatz zu einer Aktivität bei Q_1 durch das existence$(A, 1)$ Constraint). Dies ist auf folgende Beobachtung zurückzuführen: Zählt man alle verschiedenen Möglichkeiten für eine Sequenz der Länge n, die bei Q_1 möglich sind, erhält man (ohne Berücksichtigung der Reihenfolge) $(n-1)^2$, da eine Position (nämlich das A) vorgegeben ist und für alle weiteren Positionen zwei Möglichkeiten (nämlich A und C) in Betracht kommen. Für Q_2 erhält man $(n-2)^3$ verschiedene Sequenzen, da zwei Positionen (zweimal B) vorgegeben sind, dafür aber bei allen anderen Positionen drei Möglichkeiten (A, B und C) bestehen. Durch

vollständige Induktion über n [103] lässt sich leicht zeigen, dass für $n \geq 5$ gilt, dass

$$(n-2)^3 > (n-1)^2.$$

Daher besitzt Q_2 mehr gültige Sequenzen der Länge $n \geq 5$ als Q_1. Obwohl obige Ungleichung auch für $n = 4$ nicht gilt, ist dies dennoch für die zugehörigen 4-Dichten zutreffend. Dies liegt daran, dass die Berücksichtigung der Reihenfolge der Aktivitäten (diese wurde für obige Beobachtungen ignoriert) hier im Verhältnis zur Sequenzlänge noch ins Gewicht fällt. Ab $n = 5$ ist der Unterschied der Terme so groß, dass die Reihenfolge nahezu irrelevant ist. Es lässt sich also für $n \geq 4$ festhalten:

$$\lambda_n(Q_1) < \lambda_n(Q_2)$$

Alle Aussagen zur n-Dichte können aus den in Kapitel 5.2 beschriebenen Gründen für die 0-n-Dichte übernommen werden. Analog sieht man auch hier, dass $\lambda_0^n(Q_1) \geq \lambda_0^n(Q_2)$ für $n \leq 3$ und $\lambda_0^n(Q_1) \leq \lambda_0^n(Q_2)$ für $n \geq 4$.

Es fällt auf, dass alle Werte $\Lambda_n(Q_1, Q_2)$ und $\Lambda_n(Q_2, Q_1)$ der n-Ähnlichkeiten (und folglich auch alle Werte $\Lambda_0^n(Q_1, Q_2)$ und $\Lambda_0^n(Q_2, Q_1)$ der 0-n-Ähnlichkeiten) für alle $n \in \mathbb{N}$ Null sind. Das bedeutet, dass Q_1 und Q_2 keine einzige gemeinsame Sequenz akzeptieren. Dies ist aus folgendem Grund plausibel: Durch das Zusammenspiel des existence$(A, 1)$ Constraints und des exclusiveChoice(A, B) Constraints wird die Ausführung von Aktivität B in Prozessmodell Q_1 verboten. Allerdings ist die Ausführung von Aktivität B bei Prozesmodell Q_2 aufgrund des existence$(B, 2)$ Constraints mindestens zweimal notwendig. Somit können beide Prozessmodelle keine gemeinsame Sequenz besitzen, und daher sind alle n-Ähnlichkeiten (und somit auch alle 0-n-Ähnlichkeiten) Null.

Zur 0-n-Damerau-Levenshtein-Ähnlichkeit sei zunächst bemerkt, dass diese für $n = 0$ und $n = 1$ in diesem Falle nicht definiert ist, da Modell Q_2 keine Sequenz dieser Längen akzeptiert. Q_1 akzeptiert $\langle AA \rangle, \langle AC \rangle$ und $\langle CA \rangle$ der Länge $n = 2$. Offensichtlich beträgt die Damerau-Levenshtein-Distanz all dieser Sequenzen zur Sequenz $\langle BB \rangle$ (was die einzige akzeptierte Sequenz der Länge $n = 2$ von Q_2 ist) 2, da zwei Vertauschungen notwendig sind, um eine von Q_1 akzeptierte Sequenz in $\langle BB \rangle$ zu transformieren. Folglich beträgt die inverse Damerau-Levenshtein-Distanz 0, was also insgesamt eine 0-2-Damerau-Levenshtein-Ähnlichkeit von Null impliziert.

Ab $n = 3$ erkennt man, dass die 0-n-Damerau-Levenshtein-Ähnlichkeit in der Tat nicht symmetrisch ist, d.h. $\Gamma_0^n(Q_1, Q_2) \neq \Gamma_0^n(Q_2, Q_1)$ (vgl. Kapitel 5.5). Außerdem lässt sich beobachten, dass beide 0-n-Damerau-Levenshtein-Ähnlichkeiten mit steigenden n größer werden (d.h. streng monoton wachsend sind) und für $n = 3, 4, ..., 7$ die Ungleichung $\Gamma_0^n(Q_1, Q_2) > \Gamma_0^n(Q_2, Q_1)$ gilt. Diese Zusammenhänge werden im Folgenden erörtert und interpretiert. Der Grund dafür liegt wieder im Zusammenspiel der Constraints existence$(A, 1)$ und exclusiveChoice(A, B) von Q_1,

durch das eine Ausführung der Aktivität B verboten wird. Somit bestehen alle von Q_1 akzeptierten Sequenzen ausschließlich aus As und Cs. Die einzige Bedingung an eine Sequenz von Q_2 ist, dass sie mindestens zwei Bs enthält (existence$(B, 2)$). Die Folge ist, dass zur Transformation einer Sequenz t_1 von Q_1 zu einer Sequenz t_2 von Q_2 genau zwei Umformungen notwendig sind: Zwei beliebige Zeichen müssen durch ein B ersetzt werden. Folglich beträgt die inverse Damerau-Levenshtein-Distanz für eine Sequenz t_1 von Q_1 der Länge ≥ 3 zu einer Sequenz von Q_2:

$$1 - \frac{2}{\max(|t_1|, |t_2|)}$$

Man sieht, dass die inverse Damerau-Levenshtein-Distanz für größere Sequenzlängen größer wird. So lässt sich erklären, warum auch die Damerau-Levenshtein-Distanz $\Gamma_0^n(Q_1, Q_2)$ streng monoton wachsend ist. Analog kann gefolgert werden, dass die Damerau-Levenshtein-Distanz $\Gamma_0^n(Q_2, Q_1)$ streng monoton wachsend ist, da man beispielsweise aus einer Sequenz t_2 von Q_2 eine Sequenz von Q_1 erhält, indem man alle Bs durch As ersetzt. Es sind daher immer b Schritte zur Transformation notwendig, wobei b die Anzahl an Bs in t_2 bezeichnet. Aufgrund obiger Argumente ist die Damerau-Levenshtein-Distanz $\Gamma_0^n(Q_1, Q_2)$ ebenfalls streng monoton wachsend. Die Ungleichung $\Gamma_0^n(Q_1, Q_2) > \Gamma_0^n(Q_2, Q_1)$ folgt aus der Beobachtung, dass für die Transformation einer Sequenz t_2 von Q_2 in eine Sequenz t_1 von Q_1 mindestens zwei Operationen notwendig sind (da t_1 mindestens zwei Bs enthält), wohingegen umgekehrt genau zwei Operationen benötigt werden (siehe oben). Da es ab der Sequenzlänge drei mindestens eine Sequenz (nämlich $\langle BBB \rangle$) mit mehr als zwei Bs gibt, folgt, dass $\Gamma_0^n(Q_1, Q_2) > \Gamma_0^n(Q_2, Q_1)$. So lässt sich festhalten, dass eine Anpassung von Modell Q_1 an Q_2 vermutlich leichter umzusetzen wäre als umgekehrt.

Durch die detaillierte Analyse der n-Dichten und der 0-n-Damerau-Levenshtein-Ähnlichkeiten der Prozessmodelle Q_1 und Q_2 wurden bedeutende Erkenntnisse gewonnen. Die Untersuchung legt nahe, dass die Restriktivität von Q_1 im Vergleich zu Q_2 bei geringen Sequenzlängen (bis $n = 3$) zu einer höheren Dichte führt. Dies resultiert aus den „strukturierenden“ Constraints von Q_1, während Q_2 mehr Freiheiten in der Ausführung gewährt.

Interessanterweise zeigt sich jedoch ab $n = 4$, dass die Freiheiten von Q_2 zu einer höheren Anzahl gültiger Sequenzen führen, was die 0-n-Damerau-Levenshtein-Ähnlichkeiten verdeutlichen. Dies unterstreicht die Flexibilität von Q_2 bei der Modellierung von Abläufen. Die asymmetrische Natur der 0-n-Damerau-Levenshtein-Ähnlichkeiten ab $n = 3$ betont, dass die Transformation von Q_1 zu Q_2 potenziell einfacher ist als umgekehrt.

Diese Erkenntnisse weisen darauf hin, dass die Wahl zwischen den beiden Prozessmodellen von der spezifischen Anforderung an die Restriktivität und Flexibilität des Modells abhängt. Die vorgestellte Methodik bietet somit eine vielversprechende Herangehensweise zur Bewertung und Auswahl von Prozessmodellen unter Berücksichtigung unterschiedlicher Restriktionen und Freiheitsgrade.

	Helpdesk	BPIC12	BPIC13
#Sequenzen	4580	13087	1487
#Ereignisse	21348	262200	6660
#Aktivitäten	14	24	4
Durchschnittliche Sequenzlänge	4.66	20.04	4.48
Maximale Sequenzlänge	15	175	35
Minimale Sequenzlänge	2	3	1
#verschiedene Sequenzen	226	4366	183

Tabelle 8.4: Eigenschaften der verwendeten Event Logs [22]

8.3.2 Evaluation der Maße auf realen Prozessen

Im vorhergehenden Abschnitt werden die Ähnlichkeitsmaße für Prozessmodelle (Kapitel 5) auf verhältnismäßig kleinen Beispielprozessmodellen evaluiert. In diesem Kapitel sollen die Ähnlichkeitsmaße auf reale Datensätze angewendet werden, um ihren praktischen Nutzen zu veranschaulichen. Mithilfe der berechneten Ähnlichkeitswerte werden anschließend Einsichten in die Funktionsweisen der beiden verwendeten Declare-Miner, nämlich des UnconstrainedMiner [55] und des DeclareMapsMiner [104], gewonnen. Mit ihrer Hilfe werden aus drei Event Logs Declare-Prozessmodelle extrahiert, die anschließend mit Hilfe der definierten Ähnlichkeitsmaße verglichen werden.

Verwendet werden die drei Event Logs („Helpdesk“, „BPIC12“ und „BPIC13“), die vom *4TU Center for Research Data*[21] stammen. Tabelle 8.4 fasst die wichtigsten Eigenschaften der Event Logs zusammen [22]. Des Weiteren sind alle extrahierten Declare-Modelle auf GitHub[22] verfügbar.

Die beiden Miner werden so konfiguriert, dass sie genau die in der vorliegenden Arbeit betrachteten Declare Templates (Tabelle 2.2 in Kapitel 2.1.4) berücksichtigen. Zudem werden für die Konfiguration der Miner zwei Parameter benötigt: *confidence* und *support*. Diese werden im Folgenden definiert und kurz erläutert [49]:

Definition 30. *Sei E ein Event Log und C ein Declare Constraint. Der Wert*

$$support(C) := \frac{|\{t \in E \mid C \text{ in } t \text{ nicht trivial erfüllt}\}|}{|E|} \in [0, 1]$$

*heißt der **Support von** C.*

Der Support eines Constraints C bezeichnet also den Prozentsatz der Sequenzen eines Event Logs E, die C nicht trivial erfüllen. Dieser Wert wird beim Process Mining benötigt, um keine

[21] `https://data.4tu.nl`, abgerufen am: 27. Dezember 2023
[22] `https://github.com/mkaep/comparing-declare-models`, abgerufen am: 27. Dezember 2023

irrelevanten Constraints zu extrahieren. Diese werden durch das Setzen eines Mindestwerts für den Support ausgeschlossen. Um die *Confidence* eines Constraints definieren zu können, muss zunächst der Support-Begriff für Aktivitäten definiert werden [49]:

Definition 31. *Sei E ein Event Log und A eine Aktivität. Der Wert*

$$support(A) := \frac{|\{t \in E \mid \text{t enthält } A\}|}{|E|} \in [0,1]$$

heißt der ***Support von*** *A.*

Der Support von A beschreibt folglich den Prozentsatz der Sequenzen eines Event Logs, die A enthalten. Jetzt kann die *Confidence* eines Constraints definiert werden:

Definition 32. *Sei E ein Event Log und C eine Aktivität. Der Wert*

$$confidence(C) := \frac{support(C)}{support(antecedent(C))} \in [0,1]$$

heißt die ***Confidence von*** *C.*

Zum Extrahieren der Prozessmodelle werden sowohl für Support als auch Confidence der Constraints eine untere Schranke von $0,9$ gesetzt. Diese Grenze wird gewählt, da kleinere Grenzen zu Prozessmodellen mit einer viel zu großen Anzahl an Constraints (mehrere tausende) führten. Eine derartig große Anzahl an Constraints ist aber völlig fern von praktischen Anwendungen und realen Prozessmodellen. Außerdem sind die so extrahierten Prozessmodelle teilweise so restriktiv, dass sie keine einzige Ausführungssequenz erlaubten [22].

Tabelle 8.5 zeigt für jedes extrahierte Prozessmodell die Anzahl der enthaltenen Constraints [22]. Man sieht, dass der UnconstrainedMiner für jeden Event Log deutlich mehr Constraints findet als der MapsMiner. Folglich birgt der UnconstrainedMiner unter Umständen ein größeres Risiko, dass das extrahierte Modell überspezifiziert ist und keine einzige Sequenz akzeptiert. Andererseits könnten die Prozessmodelle des MapsMiners unterspezifiziert sein und zu viele Sequenzen akzeptieren. Für jeden Event Log ergab der Vergleich der jeweils korrespondierenden Prozessmodelle mit den Methoden aus Kapitel 3, dass diese weder gleich noch ineinander enthalten sind. Deshalb wurden die in Kapitel 5 vorgestellten Ähnlichkeitsmaße bis zur Sequenzlänge 5 berechnet, um Gemeinsamkeiten und Unterschiede der Modelle zu diskutieren. Diese sind in den Tabellen 8.6, 8.7 und 8.8 dargestellt.

An den Werten der n-Dichte (Tabelle 8.6) sowie der 0-n-Dichte (Tabelle 8.7) sieht man einerseits, dass die extrahierten Modelle nicht überspezifiziert sind, d.h., dass beide einige Ausführungssequenzen akzeptieren. Andererseits sind die Modelle sehr restriktiv, da beide nur eine sehr kleine Anzahl (gemessen an der Gesamtzahl an möglichen Sequenzen) an Sequenzen

Event Log	#Constraints MapsMiner	#Constraints UnconstrainedMiner
Helpdesk	94	363
BPIC13	9	34
BPIC12	13	67

Tabelle 8.5: Anzahl der Constraints der extrahierten Prozessmodelle [22]

Event Log	Miner	0	1	2	3	4	5
Helpdesk	MapsMiner	$\frac{0}{1}$	$\frac{0}{14}$	$\frac{0}{14^2}$	$\frac{0}{14^3}$	$\frac{1}{14^4}$	$\frac{2}{14^5}$
Helpdesk	UncMiner	$\frac{1}{1}$	$\frac{1}{14}$	$\frac{1}{14^2}$	$\frac{0}{14^3}$	$\frac{1}{14^4}$	$\frac{7}{14^5}$
BPIC13	MapsMiner	$\frac{0}{1}$	$\frac{0}{4}$	$\frac{1}{4^2}$	$\frac{1}{4^3}$	$\frac{3}{4^4}$	$\frac{6}{4^5}$
BPIC13	UncMiner	$\frac{1}{1}$	$\frac{0}{4}$	$\frac{1}{4^2}$	$\frac{1}{4^3}$	$\frac{3}{4^4}$	$\frac{6}{4^5}$
BPIC12	MapsMiner	$\frac{0}{1}$	$\frac{0}{24}$	$\frac{1}{24^2}$	$\frac{19}{24^3}$	$\frac{362}{24^4}$	$\frac{6860}{24^5}$
BPIC12	UncMiner	$\frac{1}{1}$	$\frac{5}{24}$	$\frac{80}{24^2}$	$\frac{3654}{24^3}$	$\frac{81923}{24^4}$	$\frac{890237}{24^5}$

Tabelle 8.6: Werte der n-Dichte λ_n für $n = 0, 1, \ldots, 5$ [22]

erlauben [22]. Eine Domänenexpertin oder ein Domänenexperte verfügt über fundiertes Fachwissen und ist in der Lage einzuschätzen, ob die berechneten Werte im akzeptablen Rahmen liegen oder die Modelle oder Umständen doch zu restriktiv sind. Daher können diese Werte in derartigen Situationen enorm hilfreich sein. So kann man z.B. erkennen, dass alle vom UnconstrainedMiner extrahierten Prozessmodelle die leere Sequenz (der Länge 0) akzeptieren, wohingegen alle vom MapsMiner extrahierten Prozessmodelle diese nicht erlauben. Dementsprechend kann je nach gewolltem Verhalten eines der Modelle angepasst werden, um die leere Sequenz zu ermöglichen bzw. zu verbieten. Analoges Verfahren kann für beliebige Sequenzen angewandt werden, in deren Akzeptanzverhalten sich die Modelle grundlegend unterscheiden. Des Weiteren lässt sich folgende überraschende Beobachtung machen: Obwohl die Modelle des MapsMiners aus weniger Constraints bestehen als die des UnconstrainedMiners, sind die Modelle des MapsMiners restriktiver, da die Werte der zugehörigen n-Dichte kleiner-gleich den Werten der n-Dichte der Modelle des UnconstrainedMiners sind. Dies steht im Gegensatz zur Erwartung, dass Prozessmodelle mit mehr Constraints restriktiver sind. Diese Beobachtung lässt die Vermutung aufkommen, dass beim UnconstrainedMiner einige redundante Constraints extrahiert werden. Diese Vermutung könnte beispielsweise mit den Ansätzen aus [15, 71, 94] überprüft werden.

Zusammenfassend lässt sich feststellen, dass die definierten Ähnlichkeitsmaße ein präzises Empfinden für die Ähnlichkeit verschiedener Prozessmodelle vermitteln können. In der Regel deuten hohe Werte auf erhebliche Ähnlichkeiten hin, während niedrige Werte eher auf geringe Ähnlichkeiten hindeuten. Es ist jedoch wichtig zu beachten, dass vereinzelte Werte einzelner

Event Log	Miner	0	1	2	3	4	5
Helpdesk	MapsMiner	$\frac{0}{1}$	$\frac{0}{15}$	$\frac{0}{211}$	$\frac{0}{2955}$	$\frac{1}{41371}$	$\frac{3}{579195}$
Helpdesk	UncMiner	$\frac{0}{1}$	$\frac{1}{15}$	$\frac{2}{211}$	$\frac{2}{2955}$	$\frac{3}{41371}$	$\frac{10}{579195}$
BPIC13	MapsMiner	$\frac{1}{1}$	$\frac{1}{5}$	$\frac{2}{21}$	$\frac{3}{85}$	$\frac{6}{341}$	$\frac{12}{1365}$
BPIC13	UncMiner	$\frac{0}{1}$	$\frac{0}{5}$	$\frac{1}{21}$	$\frac{2}{85}$	$\frac{5}{341}$	$\frac{11}{1365}$
BPIC12	MapsMiner	$\frac{0}{1}$	$\frac{0}{25}$	$\frac{1}{601}$	$\frac{20}{14425}$	$\frac{382}{346201}$	$\frac{7242}{8308825}$
BPIC12	UncMiner	$\frac{1}{1}$	$\frac{6}{25}$	$\frac{86}{601}$	$\frac{3740}{14425}$	$\frac{85663}{346201}$	$\frac{975900}{8308825}$

Tabelle 8.7: Werte der 0-n-Dichte λ_0^n für $n = 0, 1, \ldots, 5$ [22]

Event Log	0	1	2	3	4	5
Helpdesk	0	0	1	1	$\frac{2}{3}$	$\frac{3}{6}$
BPIC13	0	0	0	1	1	$\frac{2}{7}$
BPIC12	0	0	0	0.3	0.4	0.3

Tabelle 8.8: Werte der Damerau-Levenshtein-Distanz Γ_0^n für $n = 0, 1, \ldots, 5$ [22]

Maße zu Fehlinterpretationen führen können. Daher ist stets das Zusammenspiel aller Maße, das auf die jeweiligen Sequenzlängenbereiche abgestimmt ist und vom konkreten Anwendungsfall abhängt, zu berücksichtigen. Eine Möglichkeit wäre nach der in Abschnitt 8.3.1 beschriebenen Methode vorzugehen.

8.4 Theoretische Laufzeitanalyse der szenario-basierten Modellprüfung

Um die Laufzeit der szenario-basierten Modellprüfung (Kapitel 6) für ein Declare-Modell $P = (\mathcal{A}, \mathcal{T})$ zu bestimmen, werden die Einzellaufzeiten der drei Verhaltensanalysen benötigt. Verhaltensanalyse 1 (Kapitel 6.1) überprüft eine Sequenz t auf Gültigkeit. Dafür wird t im zugehörigen Prozessautomaten $M_P = (\mathcal{A}, Q, q_0, \delta, F)$ simuliert (Algorithmus 7) und danach überprüft, ob M_P die Sequenz t akzeptiert, d.h. ob $\delta(q_0, t) \in F$. Dafür müssen genau $|t|$ Schritte berechnet werden, da für jedes Symbol von t ein Schritt im Prozessautomaten M_P berechnet werden muss. Folglich hängt die Laufzeit linear (d.h. $\mathcal{O}(|t|)$) von der Länge der Sequenz ab [18].

Für Verhaltensanalyse 2 (*Vervollständigung von Sequenzen*) aus Kapitel 6.2 werden die Laufzeiten von drei aufeinanderfolgenden Schritten bestimmt [18]:

(i) Laufe im Prozessautomaten M_P zum aktuellen Zustand $\sigma(q_0, t)$.

	Laufzeit
Verhaltensanalyse 1	$\mathcal{O}(\|t\|)$
Verhaltensanalyse 2	$\mathcal{O}(\|t\|) + \mathcal{O}(\|V\| + \|E\|) + \mathcal{O}(\|\mathcal{A}\|^{2\|V\|}) = \mathcal{O}(\|\mathcal{A}\|^{2\|V\|})$
Verhaltensanalyse 3	$\mathcal{O}(\|t\|) + \mathcal{O}(\|V\| + \|E\|) + \mathcal{O}(\|\mathcal{A}\|^{l}) = \mathcal{O}(\|\mathcal{A}\|^{l})$

Tabelle 8.9: Theoretische Laufzeitanalyse der drei Verhaltensanalysen in Abhängigkeit von einer Sequenz t, von dem zum Prozessautomaten $P = (\mathcal{A}, \mathcal{T})$ gehörigen Graphen $G = (V, E)$ sowie von einer gewählten Sequenzlänge l

(ii) Finde alle Pfade minimaler Länge von $\sigma(q_0, t)$ in einen akzeptierenden Zustand $q \in F$.

(iii) Konstruiere alle Wörter entlang der gefundenen Pfade.

Schritt *(i)* erfolgt analog zu Verhaltensanalyse 1 und hat demnach eine lineare Laufzeit von $\mathcal{O}(|t|)$, wobei $|t|$ die Länge der Sequenz t bezeichnet. Für den zweiten Schritt *(ii)* wird der zu M_P gehörende Graph $G = (V, E)$ (Kapitel 6.2) mit Hilfe einer Breitensuche [75] traversiert. Diese besitzt eine Laufzeit von $\mathcal{O}(|V| + |E|)$ [75]. Im dritten Schritt *(iii)* werden alle Wörter entlang der im zweiten Schritt *(ii)* gefundenen Pfade konstruiert. Im Worst-Case-Szenario ist der komplette Graph G zu traversieren und in jedem Schritt ist jede Aktivität $A \in \mathcal{A}$ möglich (wie etwa in einem Prozessmodell, in dem alles erlaubt ist). Folglich sind in diesem Fall $|\mathcal{A}|^{2|V|}$ verschiedene Fortsetzungen von t möglich. Da diese Verhaltensanalyse vermeidet, dass Schleifen (d.h. reflexive Kanten) maximal einmal durchlaufen werden (Kapitel 6.2), ist die maximale Sequenzlänge auf $2 \cdot |V|$ limitiert. Daraus kann gefolgert werden, dass Schritt *(iii)* eine exponentielle Laufzeit besitzt, was konsequenterweise eine exponentielle Laufzeit von Verhaltensanalyse 2 zur Konsequenz hat [18].

Verhaltensanalyse 3 (*Vervollständigung von Sequenzen innerhalb von n Schritten*) aus Kapitel 6.3 unterscheidet sich von Verhaltensanalyse 2 lediglich in der Konstruktion der akzeptierten Wörter aus den gefundenen Pfaden. Wie oben gesehen, ist bei Verhaltensanalyse 2 die Länge auf $2 \cdot |V|$ begrenzt, wobei bei Verhaltensanalyse 3 die Länge frei durch die Benutzerin oder den Benutzer gewählt werden kann. Somit ergeben sich für eine gewisse Länge l im schlimmsten Falle $|\mathcal{A}|^l$ verschiedene Vervollständigungen [18], was bedeutet, dass die Gesamtlaufzeit exponentiell ist.

Alle theoretischen Laufzeiten der drei Verhaltensanalysen sind in Tabelle 8.9 zusammengefasst. Obwohl zwei der Verhaltensanalysen (Verhaltensanalyse 2 und Verhaltensanalyse 3) exponentielle Laufzeiten aufweisen, sind die zugehörigen Berechnungen dennoch für die meisten realen Anwendungen praktikabel und liefern in den meisten Fällen sogar Ergebnisse im Millisekundenbereich, da sowohl die Sequenzlängen als auch die Größe der resultierenden Automaten im kontrollierbaren Bereich bleiben. Für genauere Details sei auf Kapitel 8.5 verwiesen, wo mit Hilfe einiger realen Modelle die Verhaltensanalysen empirisch evaluiert werden.

Event Log	#Constraints	#Aktivitäten	#Zustände	#nicht-tot-Transitionen
BPIC 2017	154	9	14	20
BPIC 2018	178	10	21	55
BPIC 2019	14	4	5	8
BPIC'20(domestic)	32	5	7	5
BPIC'20(international)	116	9	11	9
BPIC'20(travelpermit)	70	9	13	37
BPIC'20(prepaid)	55	6	1	0
BPIC'20(request)	32	5	7	6

Tabelle 8.10: Übersicht über Eigenschaften der zu den betrachteten Event Logs extrahierten Declare-Modelle sowie korrespondierenden Prozessautomaten [18]

8.5 Empirische Evaluation der szenario-basierten Modellprüfung

Um die praktische Anwendbarkeit des entwickelten Ansatzes zu demonstrieren, werden in diesem Kapitel zusätzlich zur theoretischen Laufzeitanalyse (Kapitel 8.4) empirische Rechnungen auf realen Event Logs durchgeführt. Dafür wurden die Event Logs der letzten vier Process Intelligence Challenges (BPIC)[23], d.h. *BPIC 2017 (financial industry)*, *BPIC 2018 (fund process)*, *BPIC 2019 (purchase process)* und *BPIC'20 (reimbursement process)* verwendet. Der Event Log *BPIC'20 (reimbursement process)* besteht dabei aus fünf Sublogs: *BPIC'20(domestic)*, *BPIC'20(international)*, *BPIC'20(travelpermit)*, *BPIC'20(prepaid)* und *BPIC'20(request)*. Die zugehörigen Datensätze wurden vom *4TU Center for Research Data*[24] entnommen.

Zunächst wurden mit Hilfe des MINERful Miners [101, 102] aus den Event Logs wie in Kapitel 8.2 Declare Prozessmodelle extrahiert. Alle Mining-Parameter wurden dabei genauso gewählt. Die zugehörigen extrahierten Prozessmodelle sind auf GitHub[25] zu finden.

Tabelle 8.10 zeigt die Größen des zum jeweiligen Event Log extrahierten Declare-Modells $P = (\mathcal{A}, \mathcal{T})$ sowie des zugehörenden Prozessautomaten $M_P = (\mathcal{A}, Q, q_0, \delta, F)$. Die Spalten #Constraints sowie #Aktivitäten geben dabei die Anzahl der Constraints sowie der verschiedenen Aktivitäten im Declare-Modell an. Die beiden Spalten #Zustände und #nicht-tot-Transitionen beziehen sich auf den zugehörigen Prozessautomaten und geben die zugehörige Anzahl an Zuständen und die Anzahl an „nicht toten Transitionen" an. Als „nicht tote Transitionen" werden in dieser Arbeit alle Transitionen bezeichnet, die nicht in den Totzustand q_d (Kapitel 6.2) führen, also $\{\sigma(q, a) \mid \sigma(q, a) \neq q_d, q \in F \wedge a \in \mathcal{A}\}$. Alle Prozessautomaten sind ebenfalls online[25] verfügbar.

Zur Illustration des Vorgehens wird beispielhaft der *BPIC 2019* Event Log betrachtet.

[23] `https://www.tf-pm.org/competitions-awards/bpi-challenge`, abgerufen am: 27. Dezember 2023

[24] `https://data.4tu.nl`, abgerufen am: 27. Dezember 2023

[25] `https://github.com/NicolaiSchuetzenmeier/Scenario-based-Model-Checking`, abgerufen am: 27. Dezember 2023

absence(RecordGoodsReceipt, 2)
absence(CreatePurchaseOrderItem, 2)
existence(CreatePurchaseOrderItem, 1)
existence(RecordGoodsReceipt, 1)
respondedExistence(Vendorcreatesinvoice, CreatePurchaseOrderItem)
respondedExistence(Vendorcreatesinvoice, RecordGoodsReceipt)
respondedExistence(Vendorcreatesinvoice, RecordGoodsReceipt)
precedence(CreatePurchaseOrderItem, RecordGoodsReceipt)
precedence(CreatePurchaseOrderItem, RecordInvoiceReceipt)
alternateResponse(CreatePurchaseOrderItem, RecordGoodsReceipt)
notSuccession(RecordGoodsReceipt, CreatePurchaseOrderItem)
notChainSuccession(Vendorcreatesinvoice, CreatePurchaseOrderItem)
notChainSuccession(CreatePurchaseOrderItem, RecordInvoiceReceipt)
notSuccession(RecordInvoiceReceipt, CreatePurchaseOrderItem)

Tabelle 8.11: Menge $\mathcal{T}_{2019}$ der extrahierten Constraints für den *BPIC 2019* Event Log

Dieser lieferte ein für anschauliche Zwecke geeignetes Prozessmodell sowie einen überschaubaren Prozessautomaten. Die Prozessmodelle zu den anderen Event Logs wären alleine schon auf Grund der hohen Anzahl an Constraints sehr unübersichtlich.

Das zu Event Log *BPIC 2019* extrahierte Prozessmodell $P_{2019} = (\mathcal{A}_{2019}, \mathcal{T}_{2019})$ besitzt die vier Aktivitäten $\mathcal{A} = \{$Vendorcreatesinvoice, CreatePurchaseOrderItem, RecordGoodsReceipt, RecordInvoiceReceipt$\}$ und die 14 Constraints, die in Tabelle 8.11 aufgelistet sind. Abbildung 8.5 zeigt den zugehörigen Prozessautomaten zu P_{2019}. Dabei sind die Namen der Aktivitäten aus Übersichtsgründen folgendermaßen abgekürzt: Vendorcreatesinvoice (A), CreatePurchaseOrderItem (B), RecordGoodsReceipt (C), RecordInvoiceReceipt (D).

Beim Untersuchen der extrahierten Declare-Modelle fällt auf, dass sowohl das Modell zu *BPIC'20(domestic)* als auch das Modell zu *BPIC'20(international)* nur jeweils eine verschiedene Sequenz erlaubt. Dies ist darauf zurückzuführen, dass die extrahierten Constraints zu strikt sind, d.h. sie verbieten fast alles. Da Prozessmodelle, die nur eine Sequenz erlauben, für weitere Untersuchungen uninteressant sind, werden diese beiden Modelle aus der Evaluation ausgeschlossen. Des Weiteren ist ersichtlich, dass das zum Event Log *BPIC'20(prepaid)* gehörende Prozessmodell inkonsistent [15, 94] ist, d.h. es akzeptiert keine einzige Sequenz. Dies ist daran erkennbar, dass der resultierende Prozessautomat lediglich aus einem Zustand besteht und keine einzige nicht-tot-Transition besitzt (Tabelle 8.10). Aus diesem Grund wird der Event Log *BPIC'20(prepaid)* ebenfalls nicht in der Evaluation berücksichtigt. Zur Evaluation der szenario-basierten Modellprüfung wurden auf den zu den extrahierten Prozessmodellen

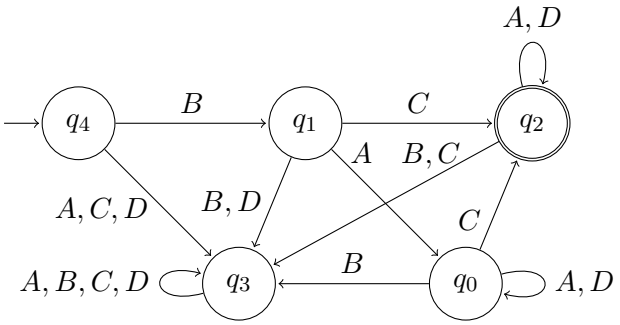


Abbildung 8.5: Prozessautomat zu P_{2019}

korrespondierenden Prozessautomaten die drei in Kapitel 6 vorgestellten Verhaltensanalysen berechnet. Alle Berechnungen wurden auf einem Windows 10 64 Bit System, das mit einem Intel Core i7-7500U CPU @2.70GHZ Prozessor, 32GB Arbeitsspeicher und SSD ausgestattet ist, durchgeführt. Um zuverlässige Laufzeiten zu garantieren, wurde sichergestellt, dass parallel keine weiteren Berechnungen auf dem gleichen Rechner laufen.

Für Verhaltensanalyse 1 wurden beliebige Sequenzen bis Länge $n = 20$ auf Akzeptanz überprüft. Diese Maximallänge wurde gewählt, da die durchschnittliche Sequenzlänge in allen betrachteten Event Logs deutlich kleiner ist. Somit sind alle relevanten Fälle mit dieser Sequenzlänge abgedeckt. Die Laufzeiten für Verhaltensanalyse 1 liegen bei allen Event Logs und allen Sequenzen im Millisekundenbereich. Dies ist genauso zu erwarten, da jeweils nur eine Sequenz im zugehörigen Prozessautomaten simuliert werden muss und weil die theoretische Laufzeit von Verhaltensanalyse 1 linear ist (Kapitel 8.4).

Verhaltensanalyse 2 weist ebenso Laufzeiten von Millisekunden auf. Dies ist auch durchaus naheliegend, da die kürzesten Vervollständigungen gesucht werden und infolgedessen nur Sequenzen entlang der kürzesten Pfade simuliert werden mussten. Man erkennt, dass die theoretisch bestimmte exponentielle Laufzeit (Kapitel 8.4) im Regelfall nicht eintritt, da ziemlich schnell valide Vervollständigungen gefunden werden.

Für die Berechnungen zu Verhaltensanalyse 3 wurde mit der leeren Sequenz ε begonnen. Das Starten mit nicht-leeren Sequenzen hätte die Laufzeit nicht beeinflusst, da sich dadurch lediglich der Zustand im Prozessautomaten, in dem der Algorithmus startet, geändert hätte. Das Berechnen des aktuellen Zustands besitzt konstante Laufzeit in Abhängigkeit von der Länge der Sequenz. Demzufolge ist die Laufzeit unabhängig von der bisherigen Teilsequenz und die leere Sequenz ε kann repräsentativ für alle Berechnungen verwendet werden. Tabelle 8.12 enthält die Laufzeiten für alle Event Logs zusammen mit den Anzahlen der berechneten Sequenzen.

Man sieht, dass fast alle Berechnungen - selbst mit Millionen von berechneten Sequenzen (z.B. *BPIC 2019*) - in wenigen Millisekunden bzw. Sekunden durchgeführt werden können.

n		*BPIC 2017*	*BPIC 2018*	*BPIC 2019*	*BPIC'20 (travelpermit)*	*BPIC'20 (request)*
10	Laufzeit	≈1	≈1	≈1	≈1	≈1
	#Sequenzen	0	0	2,304	9,140	6
12	Laufzeit	≈1	≈1	≈1	4	≈1
	#Sequenzen	13	13	11,264	235,444	8
14	Laufzeit	≈1	8	≈1	992	≈1
	#Sequenzen	153	537	53,248	5,543,860	10
16	Laufzeit	≈1	59	≈1		≈1
	#Sequenzen	867	10,951	245,760	$> 10^7$	12
18	Laufzeit	≈1	509	2		≈1
	#Sequenzen	3375	167,790	1,114,112	$> 10^7$	14
20	Laufzeit	≈1	31,923	16		≈1
	#Sequenzen	10,382	2,242,933	4,980,736	$> 10^7$	16

Tabelle 8.12: Laufzeiten (in Sekunden) und Anzahl der berechneten Sequenzen für die extrahierten Prozessmodelle in Abhängigkeit von der Sequenzlänge n [18]

Dies unterstreicht trotz der theoretischen exponentiellen Laufzeit (Kapitel 8.4) die praktische Anwendbarkeit des Ansatzes.

Die geringe Anzahl an Sequenzen bei Event Log *BPIC'20(request)* ist darauf zurückzuführen, dass das zu Grunde liegende Declare-Modell sehr restriktiv ist: Es akzeptiert keine einzige Sequenz der Länge ≤ 4 und ab der Länge $n = 5$ jeweils genau eine Sequenz der Länge n. Die relativ großen Laufzeiten für *BPIC 2018* und *BPIC'20(travelpermit)* sind darauf zurückzuführen, dass die zugehörigen Prozessmodelle sehr flexibel sind, d.h. sie erlauben eine sehr große Anzahl (mehrere Millionen) an verschiedenen Sequenzen. Diese Beobachtung wird beim Blick auf die Spalte #nicht-tot-Transitionen in Tabelle 8.10 untermauert, welche die Anzahl der Transitionen im Prozessautomaten beschreibt, die nicht in den Totzustand führen: Die Prozessautomaten zu *BPIC 2018* und *BPIC'20(travelpermit)* besitzen die größten Anzahlen an nicht-tot-Transitionen. Daher müssen hier zur Konstruktion der gültigen Sequenzen mehr Transitionen durchlaufen werden als bei den anderen Prozessautomaten. Demzufolge ist die exponentielle Laufzeit für diese beiden Event Logs „gravierender". Aus diesem Grund ist es nicht möglich, die akzeptierten Sequenzen für *BPIC'20(travelpermit)* der Länge ≥ 16 zu berechnen. Da aber die durchschnittliche Sequenzlänge im Event Log *BPIC'20(travelpermit)* deutlich geringer ($\varnothing = 12.2$) ist, sind die Berechnungen bis zur Länge 14 ausreichend und repräsentativ.

8.6 Benutzerstudie zur szenario-basierten Modellprüfung

Zur Evaluation der praktischen Anwendbarkeit der szenario-basierten Modellprüfung wurde im Oktober, November und Dezember 2022 eine Benutzerstudie durchgeführt, die in diesem Kapitel beschrieben wird [18].

Um zu verifizieren, ob ein Prozessmodell genau das gewünschte Verhalten aufweist und folglich den zu Grunde liegenden Prozess repräsentiert, muss es von der Benutzerin oder dem Benutzer vollständig verstanden werden. Gerade im Bereich der deklarativen Prozessmodellierung zeigen Studien wie z.B. [3, 17, 4, 90], dass deklarative Prozessmodelle von Menschen als sehr kompliziert empfunden werden und oftmals auch gar nicht zu überblicken und zu verstehen sind.

Die vorliegende Studie wurde entwickelt, um den Benefit der szenario-basierten Modellprüfung für deklarative Prozessmodelle (Kapitel 6) für die jeweilige Domänenexpertin bzw. den jeweiligen Domänenexperten zu illustrieren. Dabei wurde untersucht, inwieweit die drei herausgearbeiteten Verhaltensanalysen, die in einem Werkzeug umgesetzt wurden (Kapitel 7), Domänenexpertinnen und Domänenexperten dabei helfen, deklarative Prozessmodelle zu verstehen und zu verifizieren. Dafür wurde eine experimentelle Benutzerstudie entwickelt, bei der die Teilnehmerinnen und Teilnehmer Aufgaben mit und ohne Benutzung des entwickelten Werkzeugs bearbeiten sollten. Diese Aufgaben entsprechen dabei Teilschritten, die während der Verifikation oder Bearbeitung eines deklarativen Prozessmodells erledigt werden müssen.

Vom Vorgehen her wurde sich an anderen Studien [105, 106, 107, 108] orientiert, die die Verständlichkeit von Prozessmodellen aus verschiedenen Perspektiven und mit unterschiedlichen Schwerpunkten betrachten. Nach [109] ist die Verständlichkeit eines Prozessmodells eng mit seiner Nutzbarkeit verbunden, für die sich nach der ISO 9241-11-Norm zwei Schlüsselaspekte herauskristallisieren: *Effektivität* und *Effizienz*. Die *Effektivität* beschreibt dabei die Fähigkeit einer Studienteilnehmerin oder eines Studienteilnehmers, eine Aufgabe korrekt zu beantworten. Diese wird gemessen, indem die Anzahl der Fehler, die pro Aufgabe gemacht werden, erhoben wird. *Effizienz* wird über die Ressourcen (d.h. Zeit und zusätzliches Material), die der Teilnehmende benötigt, um die Aufgabe zu lösen, definiert. Zur Messung wird die benötigte Zeit gestoppt. Zudem werden die Teilnehmenden gefragt, in welchem Umfang sie das zusätzliche Material verwendet haben. Abschließend bewertet die Testperson noch ihren mental aufgebrachten Aufwand.

Inspiriert von ähnlichen Arbeiten wurden folgende drei Hypothesen aufgestellt, die in dieser Studie überprüft werden sollten [18]:

- *Hypothese 1:* Durch die Verwendung eines Werkzeugs, das die drei Verhaltensanalysen anbietet, steigt der Erfolg.
- *Hypothese 2:* Die Teilnehmenden profitieren von der Werkzeugunterstützung unabhängig von ihren jeweiligen Vorkenntnissen.

- *Hypothese 3:* Teilnehmende, die anfangs mit dem Werkzeug arbeiten, erzielen anschließend bessere Ergebnisse, auch wenn sie dann ohne das Werkzeug arbeiten, als Teilnehmende, die derartige Aufgaben zuerst ohne Werkzeug bearbeiten.

8.6.1 Studienaufbau, Materialien und Aufgabendesign

Für die Benutzerstudie wurden zwei Aufgabenblöcke S_1 und S_2 entworfen, die sowohl die gleiche Struktur als auch den gleichen Schwierigkeitsgrad besitzen. Jeder Aufgabenblock besteht dabei aus einem deklarativen Prozessmodell sowie drei zugehörigen Aufgaben 1, 2 und 3. Die Aufgaben 2 und 3 sind dabei jeweils in zwei bzw. drei unabhängige Teilaufgaben ($2a$, $2b$, $3a$, $3b$, $3c$) untergliedert.

Um zu vermeiden, dass Personen mit wenig oder gar keinen Vorkenntnissen im Prozessmanagement auf Grund von prozessspezifischen Fachbegriffen an der Bearbeitung der Aufgaben scheitern, wurden die Termini *deklaratives Prozessmodell, Constraint, Alphabet* und *Sequenz* durch die gängigeren Begriffe *Menge an Regeln, Regel, Zeichen* sowie *Zeichenkette* ersetzt. Diese Umbenennungen wurden für die Durchführung der Benutzerstudie auch im entwickelten Werkzeug angepasst. Des Weiteren wurden, um weitere Voreingenommenheiten (d.h. Fehlinterpretationen) durch eventuelles Kontextwissen zu vermeiden, als zu Grunde liegender Prozess ein abstrakter Prozess, der nicht von einem realen Prozess inspiriert ist, gewählt. Daher bestehen die Namen der Aktivitäten nur aus einem einzigen Zeichen bzw. Buchstaben. Außerdem wurde zu jeder Regel (d.h. Constraint) eine natürlichsprachige, auf Deutsch formulierte Beschreibung hinzugefügt, die die jeweilige Regel erklärt. Eine derartige Erläuterung ist notwendig, da nicht vorausgesetzt werden kann, dass die verwendeten Regeln allgemein (und vor allem mit der Materie wenig vertrauten Personen) bekannt sind. Dennoch wurden die Originalnamen der Constraints aus folgenden Gründen beibehalten [18]:

- Eine Umbenennung würde unter Umständen bei Fachleuten zu Verwirrung führen.
- Adäquate und allgemein gültige Übersetzungen sind bislang nicht vorhanden.
- Die Konsistenz mit dem entwickelten Werkzeug soll sichergestellt sein.
- Somit wird vermieden bzw. das Risiko ausgeschlossen, dass Nichtexpertinnen bzw. Nichtexperten ihnen bekannte Assoziationen anstatt der Definition der Regel zur Bearbeitung nutzen. Beispielsweise würde eine Bezeichnung wie „folgt auf“ in vielen Fällen so interpretiert werden, dass **direkt** danach das Zeichen folgen muss.
- Es soll eine Vergleichbarkeit mit anderen Studien (z.B. [17]), die ebenfalls die Originalnamen verwenden, hergestellt werden können.

Die Anweisungen und Erläuterungen zum Prozessmodell wurden den Studienteilnehmenden zu Beginn in ausgedruckter Form vorgelegt. So konnten sie während der Bearbeitung der Aufgaben jederzeit darauf zurückgreifen. Abbildung 8.6 zeigt das zu Aufgabenblock S_1 gehörige Prozessmodell. Dieses besteht aus fünf Zeichen (d.h. Aktivitäten) A, B, C, D, E und fünf Regeln (d.h. Constraints): chainResponse(A, C), succession(A, B), response(D, C), precedence(C, B) sowie notCoExistence(A, D). Außerdem ist in der letzten Spalte „Beschreibung“ die natürlichsprachige Erklärung der jeweiligen Regel enthalten. Analog zeigt Abbildung 8.7 das zu Aufgabenblock S_2 gehörende Prozessmodell. Dieses besteht ebenso aus fünf Zeichen (E, F, G, H und I) sowie vier Regeln: chainPrecedence(G, E), alternateResponse(E, F), response(F, H) sowie coExistence(G, F). Beide Prozessmodelle sind zudem online[26] verfügbar.

Verwendbare Zeichen: A, B, C, D, E

Regeln, die immer gelten müssen:

	Name	Beschreibung
Regel 1	chainResponse(A,C)	Falls das Zeichen „A“ vorkommt, muss **direkt** danach das Zeichen „C“ vorkommen.
Regel 2	succession(A,B)	Falls das Zeichen „A“ vorkommt, muss irgendwann danach das Zeichen „B“ vorkommen. Falls das Zeichen „B“ vorkommt, muss irgendwann zuvor bereits das Zeichen „A“ vorgekommen sein.
Regel 3	response(D,C)	Falls das Zeichen „D“ vorkommt, muss irgendwann danach das Zeichen „C“ vorkommen.
Regel 4	precedence (C,B)	Falls das Zeichen „B“ vorkommt, muss irgendwann zuvor bereits das Zeichen „C“ vorgekommen sein.
Regel 5	notCoExistence (A,D)	Die Zeichen „A“ und „D“ dürfen nicht gleichzeitig in der Zeichenkette vorkommen.

Abbildung 8.6: Prozessmodell zu Aufgabenblock S_1

Die Auswahl der verhältnismäßig kleinen Modellgröße (lediglich fünf Constraints) ist durch folgende drei Aspekte begründet:

[26] `https://github.com/NicolaiSchuetzenmeier/Scenario-based-Model-Checking`, abgerufen am: 27. Dezember 2023

Verwendbare Zeichen: E, F, G, H, I

Regeln, die immer gelten müssen:

	Name	Beschreibung
Regel 1	chainPrecedence(G,E)	Falls das Zeichen „E“ vorkommt, muss **direkt** zuvor das Zeichen „G“ vorgekommen sein.
Regel 2	alternateResponse(E,F)	Falls das Zeichen „E“ vorkommt, muss irgendwann danach das Zeichen „F“ vorkommen und es darf davor nicht nochmal das Zeichen „E“ vorkommen.
Regel 3	response(F,H)	Falls das Zeichen „F“ vorkommt, muss irgendwann danach das Zeichen „H“ vorkommen.
Regel 4	coExistence(G,F)	Sobald eines der Zeichen „G“ oder „F“ vorkommt, muss das jeweils andere Zeichen ebenfalls vorkommen.

Abbildung 8.7: Prozessmodell zu Aufgabenblock S_2

(i) Die in der wissenschaftlichen Literatur verwendeten Beispielprozessmodelle [21, 22, 94] sind von der Größe her vergleichbar.

(ii) Um eine hinreichende Akzeptanz sicherzustellen, sollte eine Studie in Form eines persönlichen Interviews idealerweise nicht länger als 45 Minuten dauern [110].

(iii) Eine Vorstudie ergab, dass größere Modelle von Personen mit wenig oder völlig ohne Vorwissen im deklarativen Prozessmanagement nicht mehr überblickt und verstanden werden können.

Zu jedem Prozessmodell wurden drei Aufgaben 1, 2 und 3 entwickelt. Die Aufgaben zu Aufgabenblock S_1 sind in Abbildung 8.8 abgedruckt. Die Aufgaben zu Aufgabenblock S_2 finden sich in Abbildung 8.9. Des Weiteren sind die beiden Aufgabenblöcke inklusive Lösungen auf GitHub[27] zu finden.

Alle Teilaufgaben können und sollen mit Hilfe der drei entwickelten Verhaltensanalysen (Kapitel 6) gelöst werden. Im Folgenden werden für alle Teilaufgaben der beiden Aufgabenblöcke S_1 und S_2 Lösungsvorschläge inklusive möglichem Lösungsweg präsentiert sowie die für das Lösen

[27] `https://github.com/NicolaiSchuetzenmeier/Scenario-based-Model-Checking`, abgerufen am: 27. Dezember 2023

1) Welche Regeln erfüllt die Zeichenkette „$CBEAC$“?

2a) „ACC“ ist keine gültige Zeichenkette. Wie kann „ACC“ fortgesetzt werden, damit eine minimale gültige Zeichenkette entsteht?

2b) Wie kann die Zeichenkette „CB“ fortgesetzt werden, damit eine gültige Zeichenkette entsteht?

3a) Geben Sie alle gültigen Zeichenketten bis einschließlich Länge 2 an.

3b) Geben Sie alle gültigen Zeichenketten der Länge 4 an, die mit „DE“ beginnen.

3c) Geben Sie eine gültige Zeichenkette minimaler Länge an, die zweimal das Zeichen D enthält.

Abbildung 8.8: Aufgaben zu Aufgabenblock S_1

der Aufgaben benötigten Verhaltensanalysen beschrieben. Alle Lösungen zu Aufgabenblock S_1 sind zusätzlich in kompakter Form in Abbildung 8.10 dargestellt. Eine Übersicht der Lösungen zu Aufgabenblock S_2 befindet sich in Abbildung 8.11. Außerdem gibt Tabelle 8.13 einen Überblick über die in den jeweiligen Aufgaben geforderte(n) Verhaltensanalyse(n). Diese sind für die beiden Aufgabenblöcke S_1 und S_2 jeweils identisch.

Teilaufgabe	**Beschreibung**	**Verhaltensanalyse(n)**
1	Gültigkeit von Sequenzen	1
2a	Vervollständigung von Sequenzen	2, (3)
2b	Vervollständigung von Sequenzen	2
3a	Vervollständigung innerhalb vorgegebener Länge	3
3b	Vervollständigung innerhalb vorgegebener Länge	3
3c	Vervollständigung innerhalb vorgegebener Länge	3, (2)

Tabelle 8.13: Teilaufgaben und dafür notwendige Verhaltensanalyse(n) [18]

In den nun folgenden Ausführungen werden Lösungsvorschläge zu den Teilaufgaben von Aufgabenblock S_1 beschrieben. Danach werden Lösungsvorschläge zu Aufgabenblock S_2 dargelegt.

Lösungsvorschlag zu Teilaufgabe 1) **aus Aufgabenblock** S_1

Um zu überprüfen, welche Regeln die Zeichenkette „$CBEAC$“ erfüllt, müssen alle fünf Regeln einzeln überprüft werden:

Regel 1: Diese Regel ist erfüllt, da nach jedem A (in diesem Fall gibt es nur eins) direkt ein C folgt.

1) Welche Regeln erfüllt die Zeichenkette „$EFGE$“?

2a) „HGE“ ist keine gültige Zeichenkette. Wie kann „HGE“ fortgesetzt werden, damit eine minimale gültige Zeichenkette entsteht?

2b) Wie kann die Zeichenkette „IHE“ fortgesetzt werden, damit eine gültige Zeichenkette entsteht?

3a) Geben Sie alle gültigen Zeichenketten bis einschließlich Länge 2 an.

3b) Geben Sie alle gültigen Zeichenketten der Länge 4 an, die mit „FG“ beginnen.

3c) Geben Sie eine gültige Zeichenkette minimaler Länge an, die zweimal das Zeichen F enthält.

Abbildung 8.9: Aufgaben zu Aufgabenblock S_2

Regel 2: Diese Regel ist nicht erfüllt, da nach dem Zeichen A kein B folgt.

Regel 3: Diese Regel ist trivialerweise erfüllt, da das Zeichen D in der Zeichenkette nicht vorkommt.

Regel 4: Diese Regel ist erfüllt, da vor dem einzigen B das Zeichen C bereits vorgekommen ist.

Regel 5: Diese Regel ist erfüllt, da nur das Zeichen A vorkommt (und nicht das Zeichen D).

Somit erfüllt die Zeichenkette „$CBEAC$“ die Regeln $1, 3, 4$ und 5.

Bei dieser Teilaufgabe ist eine Zeichenkette auf Gültigkeit bezüglich verschiedener Regeln zu überprüfen. Somit wird hierfür Verhaltensanalyse 1 („Gültigkeit von Sequenzen“) benötigt.

Lösungsvorschlag zu Teilaufgabe 2a) **aus Aufgabenblock** S_1

Um die Zeichenkette „ACC“ zu einer gültigen Zeichenkette vervollständigen zu können, müssen zunächst die verletzen Regeln identifiziert werden. Durch systematisches Überprüfen aller Regeln stellt man fest, dass lediglich Regel 2 (succession(A, B)) verletzt ist, da das Zeichen A in der Zeichenkette vorkommt, allerdings das Zeichen B nicht folgt. Somit vervollständigt man die Zeichenkette „ACC“ durch Anhängen des Zeichens B zu „$ACCB$“ und sieht beim abermaligen Überprüfen aller Regeln, dass sie nun alle erfüllt sind. Also ist die Zeichenkette „$ACCB$“ eine minimale Fortsetzung der Zeichenkette „ACC“ und somit die Lösung zu Aufgabe 2a.

Bei dieser Teilaufgabe ist eine Zeichenkette fortzusetzen, um eine minimale und gültige Zeichenkette zu erhalten. Hierfür wird also Verhaltensanalyse 2 („Vervollständigung von Sequenzen“) benötigt.

Alternativ kann auch systematisch vorgegangen werden, indem man versucht, die Zeichenkette immer sukzessive um ein Zeichen zu erweitern und sie anschließend auf Gültigkeit zu überprüfen. Dies wäre dann ein Vorgehen gemäß Verhaltensanalyse 3 („Vervollständigung innerhalb von n Schritten"). Allerdings ist eine solche Strategie nicht zu empfehlen, da unter Umständen sehr viele Möglichkeiten getestet werden müssen oder es im Falle einer nicht mehr gültig fortsetzbaren Zeichenkette gar nicht zum Erfolg führt (vgl. Teilaufgabe 2*b*).

Lösungsvorschlag zu Teilaufgabe 2*b*) **aus Aufgabenblock** S_1

Um die Zeichenkette „CB" zu einer gültigen Zeichenkette vervollständigen zu können, müssen zunächst die verletzen Regeln identifiziert werden. Durch systematisches Überprüfen aller Regeln bemerkt man, dass lediglich Regel 2 (succession(A, B)) verletzt ist. Im Gegensatz zu Teilaufgabe 2*a* ist dieses Mal allerdings der zweite Teil der Regel verletzt, d.h. vor dem Zeichen B müsste das Zeichen A vorkommen. Da aber Zeichen nur hinten angehängt werden dürfen, kann diese Regel nicht mehr erfüllt und folglich die Zeichenkette „CB" nicht mehr zu einer gültigen Zeichenkette fortgesetzt werden.

Bei dieser Teilaufgabe ist eine Zeichenkette fortzusetzen, um eine gültige Zeichenkette zu erhalten. Somit wird hierfür Verhaltensanalyse 2 („Vervollständigung von Sequenzen") benötigt.

Bemerkung: Im Gegensatz zu Teilaufgabe 2*a ist es hier nicht möglich, die Aufgabe mit Verhaltensanalyse* 3 *zu lösen, da sukzessives Anhängen von Zeichen immer nur die Ungültigkeit der aktuellen Zeichenkette liefern würde. Man würde demnach nie darauf kommen, dass auf Grund einer Regel, die nicht mehr erfüllt werden kann, die Zeichenkette nicht mehr gültig fortgesetzt werden kann.*

Lösungsvorschlag zu Teilaufgabe 3*a*) **aus Aufgabenblock** S_1

Um diese Aufgabe zu lösen, gibt es verschiedene Herangehensweisen. Ein möglicher Lösungsweg wäre beispielsweise, alle Zeichenketten bis einschließlich Länge 2 anzugeben und jede einzeln auf Gültigkeit zu überprüfen. Da dies aber eine sehr langwierige und mühevolle Prozedur ist (es gibt $5^0 + 5^1 + 5^2 = 31$ Möglichkeiten), wird in dieser Arbeit ein systematischerer Lösungsweg vorgestellt:

Es gibt eine Zeichenkette der Länge 0, nämlich die leere Zeichenkette ε. Diese ist eine gültige Zeichenkette, da sie trivialerweise gegen keine der fünf Regeln verstößt.

Als gültige Zeichenketten der Länge 1 entfallen „A" (auf Grund von Regel 1), „B" (auf Grund der Regeln 2 und 4) und „D" (wegen Regel 3), da diese jeweils noch (mindestens) ein weiteres Zeichen triggern würden. Deswegen bleiben nur die beiden Zeichenketten „C" und „E" übrig. Diese verletzen keine der fünf Regeln und sind somit gültige Zeichenketten der Länge 1.

Bei der Bestimmung aller gültigen Zeichenketten der Länge 2 bemerkt man zunächst, dass das Zeichen A durch die beiden Regeln 1 und 2 die Zeichen B und C triggern würde. Somit besitzt jede gültige Zeichenkette, die das Zeichen A enthält, mindestens die Länge 3. Folglich kann das Zeichen A in keiner gültigen Zeichenkette der Länge 2 vorkommen. Analog entfällt das Zeichen B auf Grund der Regeln 2 und 4. Für das Zeichen D erhält man zwei Einschränkungen: Es muss irgendwann danach das Zeichen C vorkommen (Regel 3) und das Zeichen A darf nicht gleichzeitig auftreten (Regel 5). Da das Zeichen A aber auf Grund von obigen Überlegungen sowieso nicht vorkommen darf, kann es also nur eine gültige Zeichenkette der Länge 2, die das Zeichen D enthält, geben: „DC“. Das Zeichen C hingegen triggert kein weiteres Zeichen und somit gibt es zusammen mit obigen Überlegungen genau drei Zeichenketten der Länge 2, die das Zeichen „C“ enthalten, nämlich „CC“, „CE“ und „EC“. Das Zeichen E taucht in keiner Regel auf und darf deshalb immer verwendet werden. Wegen der eben angestellten Überlegungen und der gefundenen Zeichenketten bleibt allerdings nur noch eine gültige Zeichenkette der Länge 2 übrig: „EE“.

Damit sind alle gültigen Zeichenketten bis einschließlich Länge 2 gefunden: „ε“, „C“, „E“, „DC“, „CC“, „CE“,„EE“.

Bei dieser Teilaufgabe ist eine Zeichenkette der Länge 0 zu einer gültigen Zeichenkette bis einschließlich Länge 2 fortzusetzen. Somit wird hierfür Verhaltensanalyse 3 („Vervollständigung innerhalb von n Schritten“ für $n = 2$) benötigt.

Lösungsvorschlag zu Teilaufgabe $3b$) **aus Aufgabenblock** S_1

Um alle gültigen Zeichenketten der Länge 4, die mit „DE“ beginnen, zu bestimmen, werden zunächst alle Regeln gesucht, die die Zeichenkette „DE“ nicht erfüllt. Dies ist lediglich Regel 3, welche nach dem Zeichen D das Zeichen C fordert. Auf Grund von Regel 5 darf das Zeichen A gar nicht in der Zeichenkette vorkommen. Somit folgt aus Regel 2, dass das Zeichen B ebenfalls nicht vorkommen darf, da dieses das Zeichen A implizieren würde. Folglich verbleiben lediglich die drei Zeichen C, D und E. Auf Grund von Regel 4 kann das Zeichen D aber nur an dritter (und nicht an vierter) Stelle stehen, da danach noch das Zeichen C kommen muss. Die Überprüfung aller Regeln ergibt, dass die Zeichenkette „$DEDC$“ eine gültige Zeichenkette ist. Da die beiden Zeichen C und E kein weiteres Zeichen triggern, können diese beliebig angeordnet werden. Eine Überprüfung aller Regeln führt zu dem Ergebnis, dass „$DEEC$“, „$DECC$“ und „$DECE$“ gültige Zeichenketten sind.

Somit gibt es insgesamt vier gültige Vervollständigungen der Zeichenkette „DE“: „$DEDC$“, „$DEEC$“, „$DECC$“ und „$DECE$“.

Bei dieser Teilaufgabe ist eine Zeichenkette der Länge 2 zu einer gültigen Zeichenkette der Länge 4 fortzusetzen. Daher wird hierfür Verhaltensanalyse 3 („Vervollständigung innerhalb von

1) chainResponse(A, C), response(D, C), precedence(C, B), notCoExistence(A, D).

2a) „$ACCB$“.

2b) „CB“ kann nicht gültig fortgesetzt werden, da die precedence(C, B) Regel nicht mehr erfüllt werden kann.

3a) leere Zeichenkette „ε, “,„C“,„E“,„DC“,„CC“,„CE“,„EC“,„EE“.

3b) „$DEDC$“, „$DEEC$“, „$DECC$“, „$DECE$“.

3c) „DDC“.

Abbildung 8.10: Lösungen zu Aufgabenblock S_1

n Schritten“ für $n = 2$) benötigt.

Lösungsvorschlag zu Teilaufgabe 3c) **aus Aufgabenblock** S_1

Die minimale Zeichenkette, die zweimal das Zeichen D enthält, ist offensichtlich die Zeichenkette „DD“. Allerdings ist „DD“ keine gültige Zeichenkette, da Regel 3 nicht erfüllt ist: Nach dem Zeichen D müsste nämlich irgendwann das Zeichen C folgen. Der naheliegendste Schritt ist demnach, das Zeichen C an „DD“ anzuhängen. Eine Überprüfung aller Regeln ergibt, dass die Zeichenkette „DDC“ nun alle Regeln erfüllt und somit eine gültige Zeichenkette, die zweimal das Zeichen D enthält, ist. Auf Grund obiger Beobachtungen ist diese Zeichenkette auch minimal und somit eine mögliche Lösung der Aufgabe.

Bemerkung: „DDC“ ist sogar die einzige Lösung der Länge 3*, da gemäß Regel* 3 *jedes D von einem C gefolgt werden muss. Somit muss das Zeichen C nach beiden Ds stehen, um Minimalität zu gewährleisten. Dies war aber in der Aufgabe nicht verlangt und es genügte, „DDC“ als eine gültige Lösung anzugeben.*

Bei dieser Teilaufgabe ist eine Zeichenkette der Länge 0 zu einer gültigen Zeichenkette, die zweimal das Zeichen D enthält, fortzusetzen. Dafür sind solange gültige Vervollständigungen der leeren Zeichenkette ε zu konstruieren, bis eine Vervollständigung zweimal das Zeichen D enthält. Hierfür wird Verhaltensanalyse 3 („Vervollständigung innerhalb von n Schritten“) benötigt. Alternativ kann in diesem konkreten Fall auch nach Verhaltensanalyse 2 („Vervollständigung von Sequenzen“) vorgegangen werden, indem die Zeichenkette „DD“ vervollständigt wird. Dieses Verfahren ist aber nicht allgemeingültig und funktioniert hier nur, da die gesuchte Zeichenkette „DDC“ mit „DD“ beginnt. In anderen Fällen würde dieses Vorgehen unter Umständen zwar gültige, aber nicht unbedingt minimale Lösungen ergeben.

Nun werden Lösungsvorschläge zu den Teilaufgaben von Aufgabenblock S_2 beschrieben. Da die Herangehensweise jeweils sehr ähnlich zu der zur Lösung der Teilaufgaben von Aufgabenblock S_1 ist, kann hier die Beschreibung deutlich kürzer ausfallen.

Lösungsvorschlag zu Teilaufgabe 1) **aus Aufgabenblock** S_2

Um zu überprüfen, welche Regeln die Zeichenkette „$EFGE$“ erfüllt, müssen alle fünf Regeln einzeln überprüft werden:

Regel 1: Diese Regel ist nicht erfüllt, da direkt vor dem ersten E kein G steht.

Regel 2: Diese Regel ist nicht erfüllt, da nach dem zweiten E kein F folgt.

Regel 3: Diese Regel ist nicht erfüllt, da nach dem Zeichen F kein H folgt.

Regel 4: Diese Regel ist erfüllt, da sowohl das Zeichen F als auch das Zeichen G vorkommt.

Somit erfüllt die Zeichenkette „$EFGE$“ lediglich Regel 4.

Lösungsvorschlag zu Teilaufgabe 2a) **aus Aufgabenblock** S_2

Um die Zeichenkette „HGE“ zu einer gültigen Zeichenkette vervollständigen zu können, müssen zunächst die verletzen Regeln identifiziert werden. Durch systematisches Überprüfen aller Regeln stellt man fest, dass die Regeln 2 und 4 verletzt sind. Beide Regeln fordern das Zeichen F und somit vervollständigt man die Zeichenkette zu „$HGEF$“. Beim abermaligen Überprüfen aller Regeln stellt man fest, dass bis auf Regel 3 alle Regeln erfüllt sind. Regel 3 fordert das Zeichen H nach dem Zeichen F. Das Anhängen von H liefert die Zeichenkette „$HGEFH$“. Eine Überprüfung aller Regeln liefert das Ergebnis, dass „$HGEFH$“ eine gültige Zeichenkette und somit eine (sogar die einzige) minimale Vervollständigung der Zeichenkette „HGE“ ist.

Lösungsvorschlag zu Teilaufgabe 2b) **aus Aufgabenblock** S_2

Um die Zeichenkette „IHE“ zu einer gültigen Zeichenkette vervollständigen zu können, müssen zunächst die verletzen Regeln identifiziert werden. Durch systematisches Überprüfen aller Regeln stellt man fest, dass lediglich Regel 1 (chainPrecedence(G, E)) verletzt ist. Daher müsste das Zeichen „G“ direkt vor dem Zeichen „E“ eingefügt werden. Da aber Zeichen nur hinten angehängt werden dürfen, kann diese Regel nicht mehr erfüllt werden. Also kann die Zeichenkette „IHE“ nicht mehr zu einer gültigen Zeichenkette fortgesetzt werden.

Lösungsvorschlag zu Teilaufgabe 3*a*) **aus Aufgabenblock** S_2

Auch hier wird wie bei Aufgabenblock S_1 ein systematischer Lösungsweg vorgestellt (im Gegensatz zum Durchprobieren aller Möglichkeiten).

Es gibt eine Zeichenkette der Länge 0, nämlich die leere Zeichenkette ε. Diese ist eine gültige Zeichenkette, da sie trivialerweise gegen keine der vier Regeln verstößt.

Als gültige Zeichenketten der Länge 1 fallen „E“ (auf Grund von Regel 1), „F“ (Regeln 3 und 4) und „G“ (Regel 4) weg, da diese jeweils noch (mindestens) ein weiteres Zeichen triggern würden. Es bleiben also nur die beiden Zeichenkette „H“ und „I“ übrig. Diese verletzen keine der vier Regeln und sind somit gültige Zeichenketten der Länge 1.

Beim Bestimmen aller gültigen Zeichenketten der Länge 2 bemerkt man zunächst, dass das Zeichen E durch die Regeln 1 und 2 das Zeichen G und das Zeichen F triggern würde. Es besitzt also jede gültige Zeichenkette, die das Zeichen E enthält, mindestens die Länge 3. Folglich kann das Zeichen E in keiner gültigen Zeichenkette der Länge 2 vorkommen. Analog würde das Zeichen F auf Grund der Regeln 3 und 4 die beiden Zeichen H und G triggern. Somit kann auch das Zeichen F in keiner gültigen Zeichenkette der Länge 2 auftreten. Des Weiteren wird auf Grund von Regel 4 durch das Zeichen G das Zeichen F gefordert. Da das Zeichen F aber, wie oben beschrieben, das Zeichen H triggert, kann auch das Zeichen G in keiner gültigen Zeichenkette der Länge 2 vorkommen. Somit bleiben nur die beiden Zeichen H und I, für die keine direkten Einschränkungen durch eine Regel gelten, übrig. Diese können beliebig kombiniert werden und es ergeben sich die folgenden Möglichkeiten: „HH“, „HI“, „IH“,„II“.

Damit sind alle gültigen Zeichenketten bis einschließlich Länge 2 gefunden: „ε“, „I“, „H“, „HH“, „HI“, „IH“,„II“.

Lösungsvorschlag zu Teilaufgabe 3*b*) **aus Aufgabenblock** S_2

Um alle gültigen Zeichenketten der Länge 4, die mit „FG“ beginnen, zu bestimmen, werden zunächst alle Regeln gesucht, die die Zeichenkette „FG“ nicht erfüllt. Dies ist lediglich Regel 3, die nach dem Zeichen F das Zeichen H fordert. Durch systematisches Durchprobieren findet man schließlich die folgenden sechs gültigen Fortsetzungen der Länge 4: „$FGHH$“, „$FGHI$“, „$FGHG$“, „$FGFH$“,„$FGIH$“ und „$FGGH$“.

Lösungsvorschlag zu Teilaufgabe 3*c*) **aus Aufgabenblock** S_2

Die minimale Zeichenkette, die zweimal das Zeichen F enthält, ist offensichtlich die Zeichenkette „FF“. Allerdings ist „FF“ keine gültige Zeichenkette, da die Regeln 3 und 4 nicht erfüllt sind: Nach dem Zeichen F müsste irgendwann das Zeichen H folgen (Regel 3). Außerdem müsste das Zeichen G in der Zeichenkette auftreten (Regel 4). Daher besitzt eine minimale Zeichenkette, die

1) coExistence(G, F).

2a) „$HGEFH$“.

2b) „IHE“ kann nicht gültig fortgesetzt werden, da die chainPrecedence(G, E) Regel nicht mehr erfüllt werden kann.

3a) leere Zeichenkette „ε, “,„H“,„I“,„HH“,„HI“,„IH“,„II“.

3b) „$FGHH$“, „$FGHI$“, „$FGHG$“, „$FGFH$“„$FGIH$“,„$FGGH$“,.

3c) „$FFHG$“, „$FFGH$“, „$FGFH$“.

Abbildung 8.11: Lösungen zu Aufgabenblock S_2

mindestens zweimal das Zeichen F enthält, mindestens die Länge 4. Es ist der naheliegendste Schritt, die beiden Zeichen G und H an „FF“ anzuhängen. Überprüfen aller Regeln ergibt, dass die beiden Zeichenkette „$FFHG$“ und „$FFGH$“ nun alle Regeln erfüllen und somit gültige Zeichenketten, die zweimal das Zeichen F enthalten, sind. Daher ist die minimale Länge für eine Zeichenkette mit zwei Fs vier. Man erkennt ferner, dass es nicht notwendig ist, dass die beiden Fs gleich zu Beginn stehen. So ist die Zeichenkette „$FGFH$“ ebenfalls eine gültige Zeichenkette der Länge 4. Weitere Möglichkeiten gibt es nicht, und folglich sind alle Lösungen gegeben durch: „$FFHG$“,„$FFGH$“ und „$FGFH$“.

Zielstellung und Einordnung des Werkzeug-Einsatzes

Es soll an dieser Stelle ausdrücklich betont werden, dass das primäre Ziel dieser Studie nicht darin besteht, das entwickelte Werkzeug selbst (d.h. seine Benutzerfreundlichkeit und grafische Darstellung) zu bewerten, sondern vielmehr die Nützlichkeit der identifizierten Verhaltensanalysen zu untersuchen. Dafür ist das gewählte Studiendesign ausreichend: Die Teilnehmerinnen und Teilnehmer müssen, selbst wenn sie das Werkzeug verwenden, eine Lösung für ein vorgegebenes Problem finden, indem sie jedes Mal die entsprechende(n) Verhaltensanalyse(n) identifizieren und anwenden.

Das Werkzeug selbst bietet lediglich eine Liste der relevanten Verhaltensanalysen an, was den Prozess strukturierter und zielorientierter gestaltet und somit die manuelle Arbeit erleichtert. Insgesamt dient das Werkzeug also lediglich zur Unterstützung für den Problemlösungsprozess und nimmt keinen Einfluss auf die eigentlichen Problemlösefähigkeiten der Teilnehmerinnen und Teilnehmer. Aus diesem Grund misst die Studie, inwieweit die Verhaltensanalysen tatsächlich dazu beitragen, die Verifizierung eines Prozessmodells zu unterstützen. Es soll auch noch darauf

hingewiesen werden, dass bewusst darauf verzichtet wurde, die Benutzeroberfläche des Werkzeugs zu optimieren. Auf diese Weise wird sichergestellt, dass die Ergebnisse der Studie ausschließlich auf der tatsächlichen Nützlichkeit der Verhaltensanalysen basieren und nicht durch eine verbesserte Benutzeroberfläche beeinflusst werden.

8.6.2 Studienablauf

Der Ablauf der Benutzerstudie ist in drei Phasen gegliedert:

(i) Abfrage deskriptiver Merkmale der Studienteilnehmenden.

(ii) Lösung eines Aufgabenblocks mit/ohne Werkzeugunterstützung und Bewertung des erbrachten mentalen Aufwands durch die Teilnehmenden.

(iii) Lösung des anderen Aufgabenblocks mit/ohne Werkzeugunterstützung und erneute Bewertung des mentalen Aufwands durch die Teilnehmenden.

Zunächst werden in Phase (i) in Form eines Interviews deskriptive Merkmale der Studienteilnehmenden wie z.B. Alter, Geschlecht, Bildungsstand, Beruf oder Vorkenntnisse im Bereich Prozessmanagement abgefragt. Die erfassten Daten bilden die Grundlage, um die durchgeführte Benutzerstudie auswerten und die Ergebnisse interpretieren zu können. Der für die Erfassung der Daten verwendete Fragebogen inklusive aller erfassten deskriptiven Merkmale ist in Anhang A.3 und auf GitHub[28] zu finden. Zusätzlich sind auf GitHub sämtliche von den Teilnehmenden erhobenen deskriptiven Merkmale als CSV-Datei verfügbar.

Anschließend wird ihnen eine kurze Einführung in die Thematik der Studie gegeben. Dabei wird ihnen das Prozessmodell (Kapitel 8.6.1) des zuerst zu bearbeitenden Aufgabenblocks (dieser wurde vor Beginn der Studie durch Zufall ausgewählt) in ausgedruckter Form vorgelegt, anhand dessen die wichtigsten Fachbegriffe wie *Regel, Menge von Regeln, Zeichen* sowie *Zeichenkette* (vgl. Kapitel 8.6.1) erklärt werden. Außerdem wird den Studienteilnehmenden die Möglichkeit gegeben, Fragen zu stellen. Schließlich werden die Studienteilnehmenden noch darüber informiert, dass sie während der Bearbeitung der Studie zur jedem Zeitpunkt die Hilfsmittel (d.h. die Übersicht über die Menge an Regeln und deren Erklärungen) verwenden dürfen.

Neben der zufällig bestimmten Reihenfolge der Bearbeitung der Aufgabenblöcke und der Klärung der Frage, ob zunächst mit oder ohne Werkzeugunterstützung gearbeitet wird, wird zudem die Reihenfolge der drei Aufgaben innerhalb eines Aufgabenblocks randomisiert. Diese zusätzliche Randomisierung verhindert das Auftreten von Lerneffekten innerhalb eines Aufgabenblocks. Insgesamt werden also die folgenden drei Aspekte randomisiert entschieden:

[28] `https://github.com/NicolaiSchuetzenmeier/Scenario-based-Model-Checking`, abgerufen am: 27. Dezember 2023

1. Beginn mit oder ohne Werkzeugunterstützung.
2. Beginn mit Aufgabenblock S_1 oder Aufgabenblock S_2.
3. Randomisierung der Reihenfolge der Aufgaben innerhalb der beiden Aufgabenblöcke.

Im Gegensatz zu den Aufgaben wird allerdings die Reihenfolge der Teilaufgaben innerhalb einer Aufgabe immer fix gehalten, d.h. Teilaufgabe a) wurde immer zuerst bearbeitet, dann Teilaufgabe b), dann ggf. Teilaufgabe c). Der Grund hierfür ist, dass die Teilaufgaben innerhalb einer Aufgabe nach dem Schwierigkeitsgrad geordnet sind und jeweils zunächst die leichteren Aufgaben bearbeitet werden sollen. Alle Randomisierungen bezüglich der Teilnehmenden sind online[28] in einer CSV-Datei verfügbar.

Um spätere Auswertungen zu ermöglichen, wird für jede Teilaufgabe die benötigte Zeit in Sekunden erfasst und festgehalten. Außerdem wird die Anzahl an Fehlern indirekt durch die Anzahl an Punkten, die es für jede Teilaufgabe gibt, gemessen. Dabei liegt die maximal erreichbare Punktzahl pro Teilaufgabe zwischen 2 und 8 Punkten. Die Punkte sind dabei für Aufgabenblock S_1 folgendermaßen verteilt: Teilaufgabe 1) fünf Punkte, die Teilaufgaben $2a), 2b)$ und $3c)$ jeweils zwei Punkte, Teilaufgabe $3a)$ acht Punkte und Teilaufgabe $3b)$ vier Punkte. Die Teilaufgaben von S_2 wurden folgendermaßen bepunktet: Teilaufgabe 1) vier Punkte, die Teilaufgaben $2a), 2b)$ und $3c)$ jeweils zwei Punkte, Teilaufgabe $3a)$ sieben Punkte und Teilaufgabe $3b)$ sechs Punkte.

Punkte werden dabei auf korrekte Antworten gegeben, während fehlende oder nicht bearbeitete Teile mit null Punkten bewertet werden. Falsche Antworten werden mit Punktabzug „bestraft“, allerdings kann eine Probandin oder ein Proband pro Aufgabe nicht weniger als null Punkte bekommen. Nach der Bearbeitung einer Teilaufgabe bekommt die Testperson ein kurzes Feedback vom Interviewer. Nach Abschluss eines Aufgabenblocks wird der Studienteilnehmerin oder dem Studienteilnehmer eine ausgedruckte Skala ausgehändigt, auf der sie oder er ihren bzw. seinen während der Bearbeitung des Aufgabenblocks mental aufgebrachten Aufwand zwischen 0 und 220 einschätzen soll. Diese Skala ist in der Wissenschaft üblich, um den mentalen Aufwand bei Benutzerstudien zu messen [111]. Abbildung 8.12 zeigt diese Skala.

Unmittelbar vor Beginn der Bearbeitung mit Werkzeugunterstützung wird dem Studienteilnehmenden eine kurze Einführung in das Werkzeug gegeben. Der Interviewer erklärt dabei die verschiedenen Funktionalitäten des Werkzeugs und zeigt, wie Zeichenketten eingegeben werden können, wie diese analysiert werden und wo die berechneten Resultate angezeigt werden. Zudem wird im Falle von technischen Problemen stets Hilfe vom Interviewer angeboten. Jedoch ist diese Hilfe lediglich auf technische Probleme, die die Bedienung des Werkzeugs betreffen, limitiert und schließt Fragen zu den Teilaufgaben oder deren Lösungen aus.

Aufgrund des gewählten Studiendesigns, das nicht darauf abzielt, die Nutzbarkeit des Werkzeugs, sondern die Nützlichkeit der Verhaltensanalysen zu evaluieren (Kapitel 8.6.1), beeinflusst

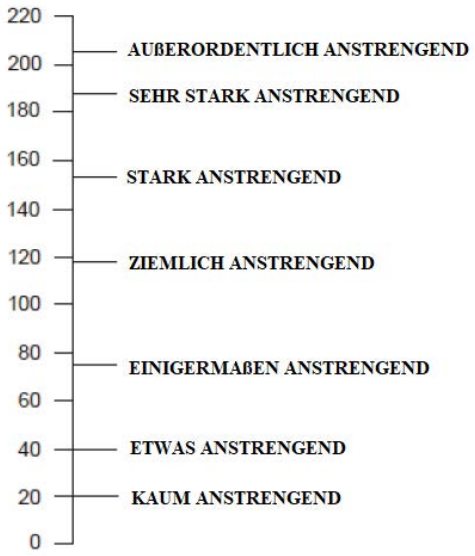


Abbildung 8.12: Skala zur Bewertung der mentalen Anstrengung bei der Bearbeitung eines Aufgabenblocks

diese Art der Hilfestellung die Ergebnisse der Benutzerstudie nicht. Nachdem die Bearbeitung des Aufgabenblocks mit Werkzeugunterstützung abgeschlossen ist, werden die Teilnehmenden gefragt, ob sie für die Bearbeitung das ausgehändigte Prozessmodell inklusive der Erklärungen der Regeln verwendet haben. Die Antwort wird schriftlich festgehalten.

8.6.3 Studienergebnisse

An der Benutzerstudie nahmen insgesamt 29 Personen teil. Von allen wurden das Alter, das Geschlecht, der Bildungsstand, die Berufserfahrung sowie ihre Vorkenntnisse im Prozessmanagement erfasst. Tabelle 8.14 zeigt die detaillierten deskriptiven Statistiken.

Im Folgenden werden die drei Hypothesen, die zu Beginn von Kapitel 8.6 gestellt wurden, überprüft. Dazu werden die erfassten Daten ausgewertet und interpretiert.

Überprüfung von Hypothese 1

Um Hypothese 1 (*„Durch die Verwendung eines Werkzeugs, das die drei Verhaltensanalysen anbietet, steigt der Erfolg.“*) zu überprüfen, werden zunächst die für die jeweiligen Aufgaben benötigten Zeiten mit und ohne Werkzeugunterstützung verglichen. Die jeweiligen Durchschnittszeiten sind in Abbildung 8.13 graphisch dargestellt. Man sieht, dass für alle Teilaufgaben - mit Ausnahme von Teilaufgabe 3c - mit Verwendung des Werkzeugs deutlich weniger Zeit benötigt wurde als ohne. Vor allem bei den komplexeren Teilaufgaben 3a und 3b, bei denen manuell eine

Variable	Wertebereich	Wert / Anzahl	Standardabweichung / Prozentsatz
Alter	Minimum / Maximum / Durchschnitt	24.0 / 69.0 / 37.21	13.93
Geschlecht	männlich / weiblich / divers	19 / 10 / 0	65.52% / 34.48% / 0.0%
höchster Schulabschluss	Grundschule / Mittelschule / Realschule / Gymnasium / Universität	0 / 0 / 4 / 3 / 22	0.0% / 0.0% / 13.79% / 10.34% / 75.86%
Berufserfahrung (in Jahren)	Minimum / Maximum / Durchschnitt	0.0 / 48.0 / 9.9	12.05
Vorkenntnisse Prozessmanagement	ja / nein	18 / 11	62.07% / 37.93%
Prozessmodelle lesen	ja / nein	17 / 1	94.44% / 5.56%
Prozessmodelle erstellen	ja / nein	11 / 7	61.11% / 38.89%
Deklarative Prozessmodelle lesen	ja / nein	6 / 12	33.33% / 66.67%
Deklarative Prozessmodelle erstellen	ja / nein	4 / 14	22.22% / 77.78%
Erfahrung im Prozessmanagement (in Jahren)	Minimum / Maximum / Durchschnitt	0.0 / 31.0 / 3.45	7.32
Erfahrung im deklarativen Prozessmanagement (in Jahren)	Minimum / Maximum / Durchschnitt	0.0 / 28.0 / 2.31	6.7

Tabelle 8.14: Demografische Merkmale und Vorkenntnisse der Studienteilnehmenden im Bereich Prozessmanagement [18]

Vielzahl an Möglichkeiten durchprobiert werden musste (Kapitel 8.6.1), ist die Zeitersparnis durch das Werkzeug immens. Das Verhalten bei Teilaufgabe 3*c*, bei der die Teilnehmenden eine Zeichenkette finden mussten, in der ein bestimmtes Zeichen genau zweimal vorkommt, ist folgendermaßen zu erklären: Da die Prozessmodelle in beiden Aufgabenblöcken das Vorkommen der gewünschten Aktivität direkt als die ersten beiden Zeichen der Zeichenkette erlauben, führte ein offensichtlicher erster Versuch unmittelbar zur korrekten Lösung. Daher konnte diese Aufgabe genauso schnell oder in einigen Fällen sogar schneller ohne Werkzeugunterstützung gelöst werden. Trotz der festgestellten allgemeinen Zeitersparnis durch Werkzeugunterstützung offenbart die Studie hier also interessanterweise auch Situationen, in denen Teilnehmende ohne dieses sogar schneller agieren können. Die Ergebnisse verdeutlichen, dass die Effizienz der Werkzeugunterstützung stark vom konkreten Anwendungsfall abhängt. Dennoch lässt sich insgesamt argumentieren, dass der Einsatz des Werkzeugs auf Grund des immensen Zeitgewinns auf jeden Fall gerechtfertigt ist.

Abbildung 8.14 zeigt die erreichte Punktzahl der Teilnehmenden, die jeweils mit bzw. ohne Werkzeugunterstützung erreicht wurde. Hier lässt sich ebenfalls feststellen, dass mit Hilfe des Werkzeugs bei allen Aufgaben mehr Punkte erreicht wurden, d.h. der Erfolg größer war als ohne. Analog zum Zeitgewinn ist der Unterschied in der Anzahl der erreichten Punkte bei den beiden Teilaufgaben 3*a* und 3*b* am gravierendsten. Dieses Resultat ist wiederum damit zu erklären,

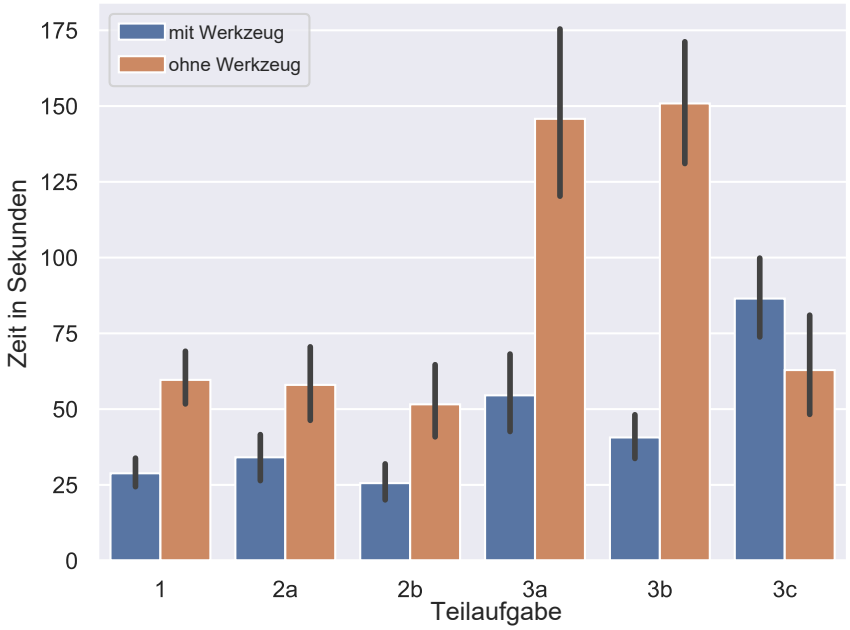


Abbildung 8.13: Benötigte Zeit für die Bearbeitung der Teilaufgaben mit und ohne Werkzeugunterstützung [18]

dass bei der Bearbeitung dieser Teilaufgaben eine Vielzahl an verschiedenen Möglichkeiten durchprobiert werden musste und somit die Verwendung des Werkzeugs zu einer erheblich größeren Zeitersparnis führt. Im Folgenden werden noch häufig produzierte Fehler und Fehlinterpretationen, die beim Durchführen der Studie immer wieder auftauchten, beschrieben:

- In Teilaufgaben, bei denen es darum ging, alle gültigen Zeichenketten bis zu einer bestimmten Länge zu finden, wurde von mehreren Teilnehmenden ein Fall übersehen – nämlich die leere Zeichenkette ε. Dies kann darauf zurückgeführt werden, dass derartige Randfälle aufgrund ihrer Abstraktheit leicht übersehen werden können oder unbeachtet bleiben, insbesondere weil ein Begriff ohne Länge in einem nichtmathematischen Kontext oft unbekannt ist. Des Weiteren konnte festgestellt werden, dass vor allem Personen mit einem mathematischen und/oder technischen Hintergrund diesen Fehler deutlich seltener machten. Auch diese Beobachtung scheint plausibel, da derartige Personen eben genau für solche Randfälle sensibilisiert sind.

- Für Teilaufgaben, bei denen alle gültigen Zeichenketten bis zu einer gewissen Länge gefunden werden sollten, fehlte einigen Studienteilnehmenden das logische Grundverständnis. Sie versuchten, alle möglichen Zeichenketten zu konstruieren und auf Gültigkeit zu prüfen. So gaben sie teilweise nach einigen Minuten frustriert auf, obwohl die Aufgaben durch geschicktes Argumentieren recht schnell zu bearbeiten gewesen wären.

- Bei zusammengesetzten Regeln wie „succession“ wurde oftmals der zweite Teil der Regel komplett ignoriert. So wurde meistens der erste Satz der Beschreibung der Regeln überprüft und danach die Überlegung eingestellt. Offensichtlich führte das Nichtbetrachten des zweiten Teils zu Fehlern.

- Einige der Studienteilnehmenden erkannten korrekterweise die Ungültigkeit gewisser Zeichenketten auf Grund von verletzten Regeln. Nach Fortsetzen der ungültigen Zeichenkette wurde aber oftmals nicht beachtet, dass zuvor geltende Regeln durch die Veränderung an der Zeichenkette potentiell verletzt wurden. Hier wurde der Zusammenhang nicht erkannt, dass Regeln zwar für eine Zeichenkette gelten können, diese aber durch Anhängen gewisser Zeichen eventuell nicht mehr gelten.

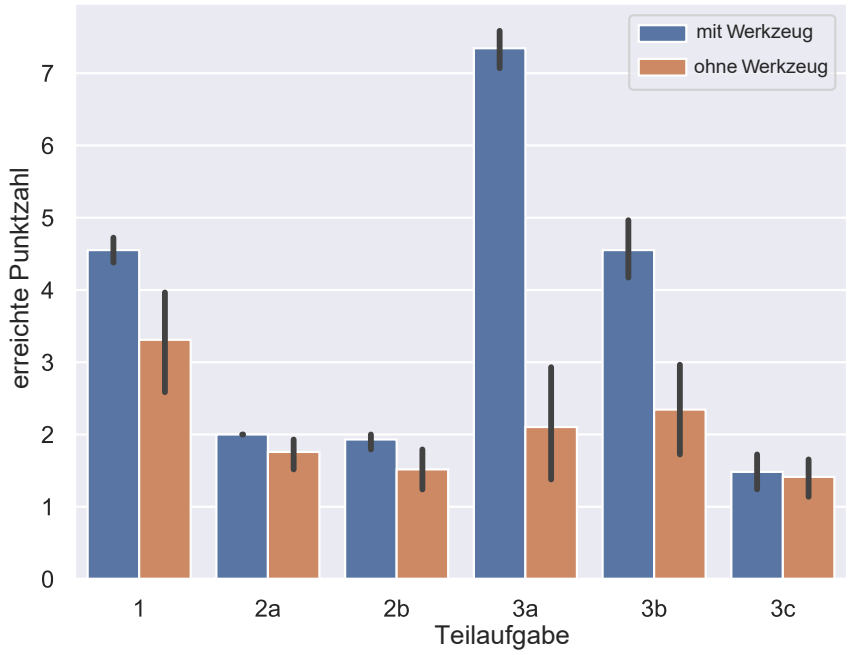


Abbildung 8.14: Erreichte Punktzahl für die Bearbeitung der Teilaufgaben mit und ohne Werkzeugunterstützung [18]

Überprüfung von Hypothese 2

Um Hypothese 2 (*„Teilnehmende profitieren von der Werkzeugunterstützung unabhängig von ihren jeweiligen Vorkenntnissen.“*) zu überprüfen, wird zunächst der von den Teilnehmenden nach der Bearbeitung der Aufgabenblöcke selbst eingeschätzte mentale Aufwand ausgewertet. Abbildung 8.15 zeigt jeweils den mentalen Aufwand mit und ohne Werkzeugunterstützung aufgeteilt nach

Vorkenntnissen. Unabhängig von den Vorkenntnissen bewerteten die Teilnehmenden den mentalen Aufwand ohne Werkzeugunterstützung als anstrengend. Wie in ähnlichen Studien [105, 112] gilt ein mentaler Aufwand von 150 (auf einer Skala von 220) bereits als große Anstrengung. Außerdem lässt sich aus Abbildung 8.15 ablesen, dass Personen ohne Vorkenntnisse im Bereich Prozessmanagement ihren mentalen Aufwand deutlich höher einstuften als Personen mit Vorkenntnissen. Bei beiden Gruppen lag der mentale Aufwand bei der Bearbeitung der Aufgaben mit Werkzeugunterstützung in einem akzeptablen Bereich (< 80). Folglich lässt sich eine deutliche Reduzierung des mentalen Aufwands durch die Werkzeugunterstützung feststellen. Ebenso sieht man, dass der mentale Aufwand durch die Werkzeugunterstützung bei Personen ohne Vorkenntnissen deutlich stärker abnimmt als bei Personen mit Vorkenntnissen. Dies lässt sich dadurch erklären, dass die mentale Anstrengung bei dieser Gruppe eher durch die vielen durchzuführenden Schritte verursacht wird und weniger durch die Lösung des Problems an sich. Außerdem kann beobachtet werden, dass 24 der 29 getesteten Personen angaben, dass sie zur Bearbeitung des Aufgabenblocks mit Werkzeugunterstützung nicht auf das ausgedruckte Aufgabenblatt zurückgriffen, welches das Prozessmodell sowie die Erklärungen der Regeln enthielt (Kapitel 8.6.1). Somit ist bei der Verwendung des Werkzeugs auch keine explizite Erklärung von Regeln mehr notwendig und die Erleichterung bei der Arbeit sowohl für Nichtexpertinnen bzw. Nichtexperten als auch Expertinnen bzw. Experten wird unterstrichen. Außerdem hat die Studie gezeigt, dass das Werkzeug völlig unabhängig von sämtlichen demographischen Merkmalen eine deutliche Unterstützung bei der Bearbeitung von Prozessmodellen ist. Beispielhaft dafür sind in Abbildung 8.16 Grafiken für das Geschlecht und den Schulabschluss abgebildet.

Insgesamt lassen sich völlig unabhängig von Vorkenntnissen bei fast allen Aufgaben erhebliche Zeitgewinne und Punktzunahmen (was eine geringere Fehlerquote impliziert) feststellen. Diese sind in Abbildung 8.17 dargestellt. Der Zeitverlust bei Teilaufgabe 3*c* lässt sich wie bereits bei Hypothese 1 damit erklären, dass auf Grund der Konstruktion des Prozessmodells bei dieser Aufgabe der erste Versuch bereits zu einem richtigen Ergebnis führte. Daher waren hier die meisten Teilnehmenden ohne Werkzeugunterstützung schneller als mit Werkzeugunterstützung.

Überprüfung von Hypothese 3

Beim Testen von Hypothese 3 (*„Teilnehmende, die anfangs mit dem Werkzeug arbeiten, erzielen anschließend bessere Ergebnisse, auch wenn sie dann ohne das Werkzeug arbeiten, als Teilnehmende, die derartige Aufgaben zuerst ohne Werkzeug bearbeiten.“*) wird festgestellt, dass mit Ausnahme der beiden Teilaufgaben 1 und 2*b* ein kleiner Lerneffekt auftritt. Das bedeutet, dass Studienteilnehmende, die mit Werkzeugunterstützung gearbeitet haben, anschließend bei der Bearbeitung der Aufgaben ohne Werkzeugunterstützung besser abgeschnitten haben als diejenigen, die zuerst ohne Werkzeugunterstützung gearbeitet haben. Ein ähnlicher Effekt ist bei der gemes-

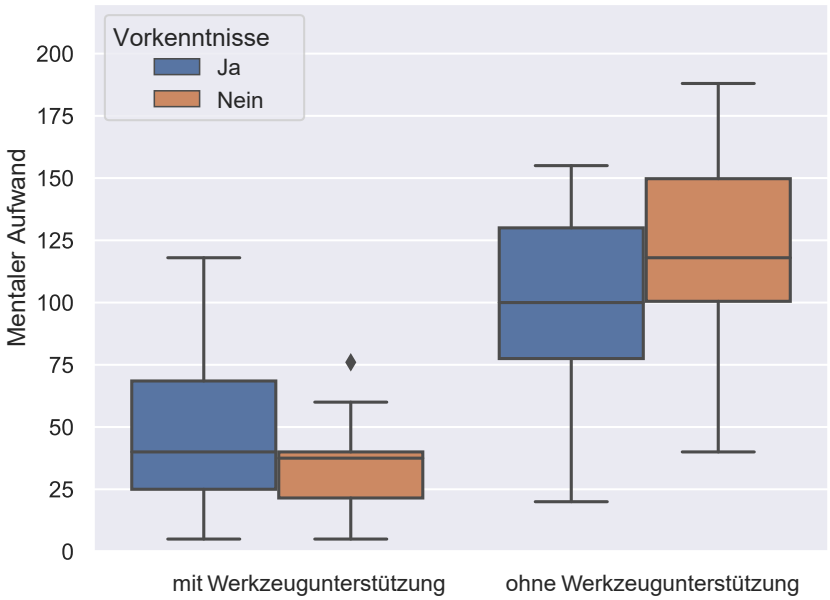


Abbildung 8.15: Mentaler Aufwand für die Bearbeitung der Aufgabenblöcke mit und ohne Werkzeugunterstützung, aufgeteilt nach Vorkenntnissen (Skala von 0 (niedrig) bis 220 (sehr hoch)) [18]

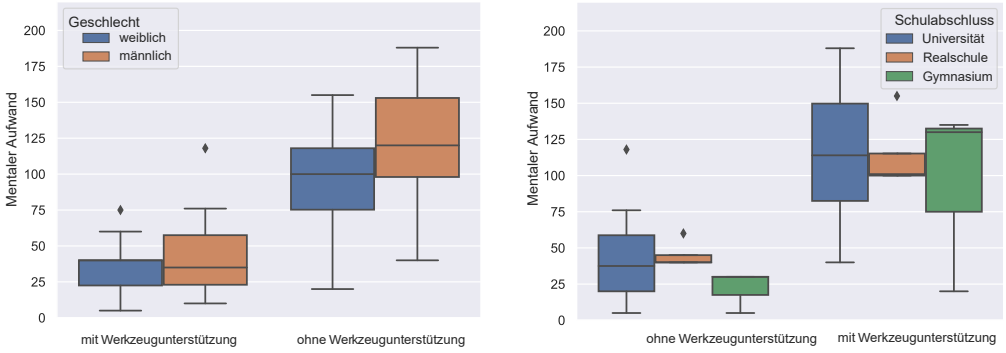


Abbildung 8.16: Reduzierung des mentalen Aufwands durch Werkzeugunterstützung in Abhängigkeit von Geschlecht (links) und Schulabschluss (rechts)

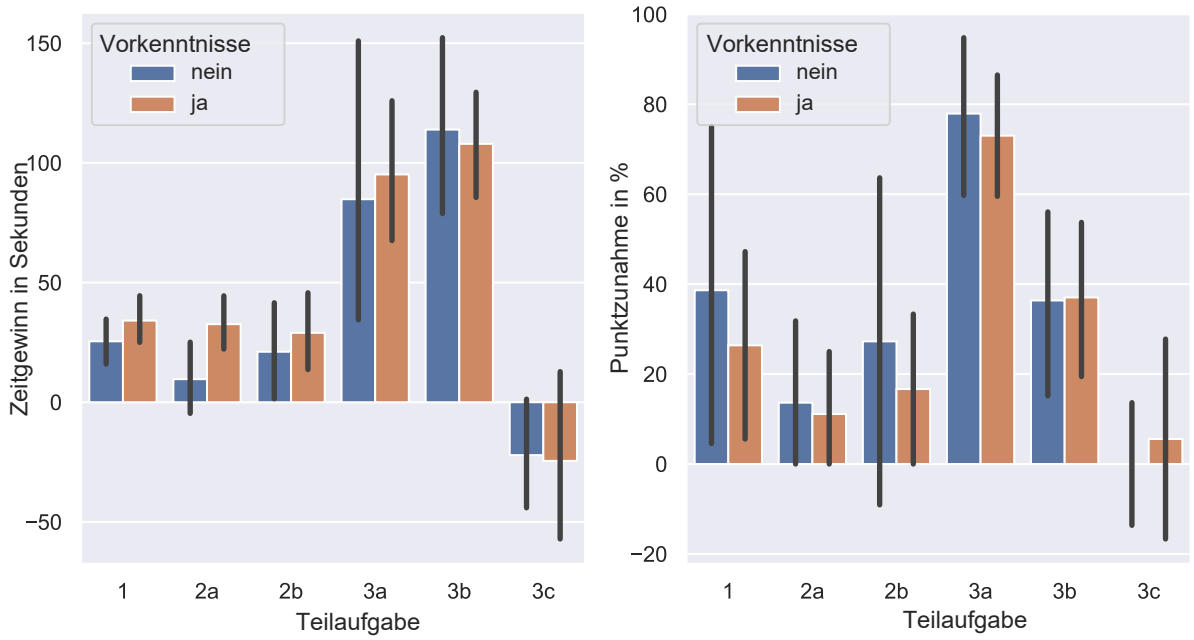


Abbildung 8.17: Zeitgewinne und Punktzunahmen durch Verwendung des Werkzeugs aufgeteilt nach Vorkenntnissen [18]

senen Zeit zu beobachten. Auch hier waren Teilnehmende (mit Ausnahme der Teilaufgabe 3*b*), die zunächst mit Werkzeugunterstützung begannen, bei der anschließenden manuellen Bearbeitung der Teilaufgaben schneller als solche, die zuerst ohne Werkzeugunterstützung arbeiteten. Diese Ergebnisse sind in Abbildung 8.18 visualisiert. Dabei bezeichnet *tool_first* die Teilnehmergruppe, die zunächst mit Werkzeugunterstützung begann und *tool_second* die Teilnehmergruppe, die die Teilaufgaben zuerst manuell, d.h. ohne Werkzeugunterstützung, bearbeitete.

Zusammenfassend lassen sich alle drei Hypothesen, die zu Beginn der Benutzerstudie aufgestellt wurden, klar bestätigen. Die Ergebnisse zeigen, dass die Verwendung des Werkzeugs zu einem deutlichen Anstieg des Erfolgs führt, indem sowohl der mentale Aufwand als auch die benötigte Zeit signifikant reduziert werden. Die positive Auswirkung der Werkzeugunterstützung erstreckt sich über verschiedene Aufgaben und bleibt unabhängig von den Vorkenntnissen der Studienteilnehmenden im Bereich Prozessmanagement bestehen. Darüber hinaus verdeutlichen die beobachteten Lerneffekte, dass die Nutzung des Werkzeugs nicht nur eine momentane Erleichterung darstellt, sondern auch einen nachhaltigen Einfluss auf die manuelle Bearbeitung der Aufgaben hat.

Diese klaren Bestätigungen stärken die Argumentation für die Integration des Werkzeugs in den Prozess der Prozessmodellierung und der Prozessanalyse erheblich. Die Ergebnisse der Benutzerstudie liefern somit nicht nur eine Bestätigung der Erwartungen, sondern unterstreichen

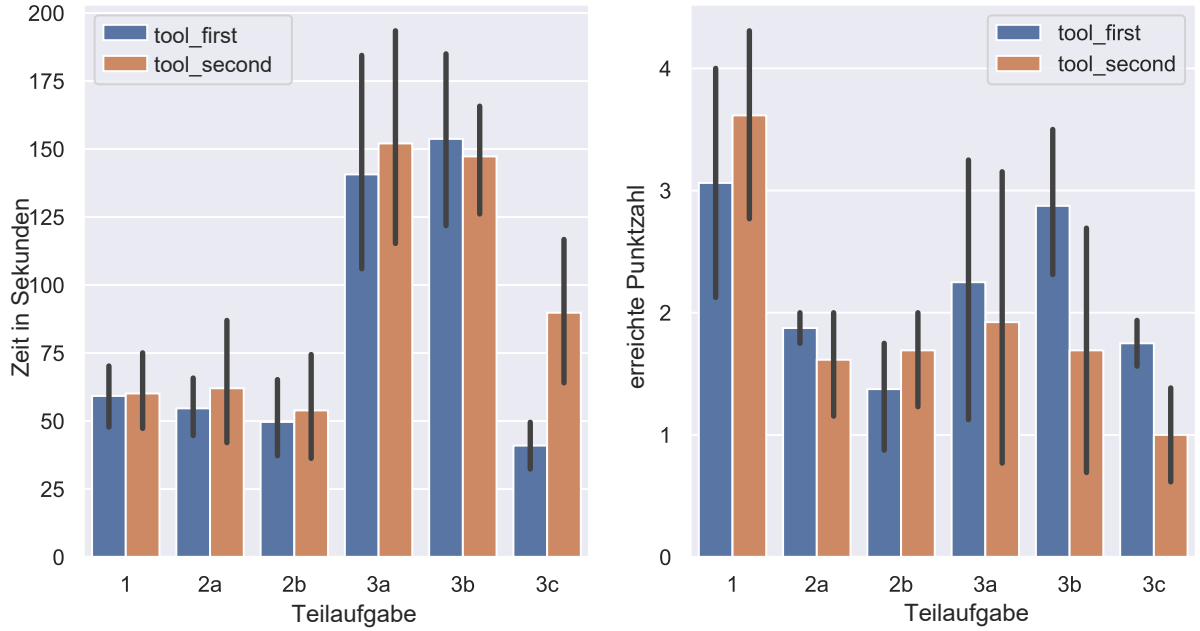


Abbildung 8.18: *Links:* Vergleich der benötigten Zeiten bei der manuellen Bearbeitung der Teilaufgaben bei Beginn mit dem Werkzeug (*tool_first*) und bei Beginn ohne Werkzeug (*tool_second*) *Rechts:* Vergleich der erreichten Punktzahlen bei der manuellen Bearbeitung der Teilaufgaben bei Beginn mit dem Werkzeug (*tool_first*) und bei Beginn ohne Werkzeug (*tool_second*) (mehr Punkte bedeuten ein besseres Ergebnis und zeigen eine geringere Fehlerquote) [18]

auch das Potenzial des Werkzeugs, eine effektive und zeitsparende Ressource für die Arbeit mit deklarativen Prozessmodellen zu sein. Die insgesamt positive Resonanz der Benutzerinnen und Benutzer deutet außerdem darauf hin, dass das Werkzeug nicht nur dazu beiträgt, die Erwartungen zu erfüllen, sondern auch Unsicherheiten und Hemmnisse, die normalerweise mit der deklarativen Modellierung verbunden sind, effektiv verringert. Diese entscheidende Erkenntnis hebt die praktische Anwendbarkeit des Werkzeugs hervor und unterstreicht dessen Fähigkeit, einen reibungslosen Übergang zur deklarativen Modellierung zu erleichtern. Somit wird nicht nur die Effizienz, sondern auch die Nutzerakzeptanz gesteigert, was für eine erfolgreiche Integration in den Prozess der Prozessgestaltung spricht.

Kapitel 9

Fazit und Ausblick

Dieses Kapitel fasst die Motivation, die Ziele, das Vorgehen und die Ergebnisse der vorliegenden Dissertation zusammen (Kapitel 9.1). Außerdem werden Einschränkungen der entwickelten Methoden und Ansätze erläutert (Kapitel 9.2). Den Abschluss bilden Vorschläge zu zukünftigen Forschungsthemen in diesem Bereich (Kapitel 9.3).

9.1 Zusammenfassung

Bei der Kooperation oder beim Zusammenschluss verschiedener Unternehmen oder Abteilungen innerhalb eines Unternehmens ist es oftmals eine große Herausforderung, die jeweiligen unternehmensspezifischen Prozesse zu analysieren und vor allem gemeinsame Elemente oder gravierende Unterschiede herauszuarbeiten. Insbesondere die Verwendung verschiedener Prozessmodellierungssprachen erschwert diese Aufgaben oder macht sie für manuelle Untersuchungen schier unmöglich. Auch das kontinuierliche Überarbeiten und Anpassen von Prozessmodellen erfordert die Anwendung von Verifikationstechniken, um sicherzustellen, dass die beabsichtigten Änderungen angemessen umgesetzt wurden. Zur Lösung dieser Probleme wurden in dieser Arbeit auf Automatentheorie basierende Methoden und Verfahren entwickelt, um gängige Problemstellungen automatisiert beantworten zu können. So werden unter anderem Verfahren präsentiert, die es ermöglichen, Prozessmodelle unabhängig von ihrer imperativen oder deklarativen Modellierung

- bezüglich sämtlicher Prozessperspektiven auf Gleichheit (Kapitel 3.2.4 und 3.2.6) und Ähnlichkeit (Kapitel 5) zu überprüfen,
- auf Gemeinsamkeiten (Kapitel 3.2.8) und Unterschiede (Kapitel 3.2.9) in deren Abläufen (d.h. bezüglich der Kontrollflussperspektive) zu untersuchen und/oder
- auf gegenseitiges Enthaltensein (Kapitel 3.2.7) zu untersuchen.

Vor allem im Bereich der deklarativen Prozessmodellierung war die Frage nach der Gleichheit oder dem gegenseitigen Enthaltensein von Prozessmodellen bisher ein offenes Problem. Dieses wird in der vorliegenden Arbeit gelöst. Insgesamt liefern die oben genannten Punkte die Antwort zu Forschungsfrage **RQ1** (*Sind zwei Prozessmodelle bezüglich ausgewählter Prozessperspektiven gleich? Die Gleichheit soll unabhängig von der gewählten Modellierungssprache bestimmt werden können.*). Man kann also festhalten, dass diese Forschungsfrage in vollem Umfang beantwortet wurde.

Aufgrund von permanenten Änderungen der realen Abläufe in Unternehmen müssen Prozessmodelle kontinuierlich angepasst und/oder verändert werden. Dabei kann oftmals manuell nicht herausgefunden werden, ob die durchgeführten Veränderungen am Modell auch die tatsächlichen Neuerungen repräsentieren. Aus diesem Grund wird in dieser Arbeit eine Methode zur Berechnung von Verhaltensänderungen von Prozessmodellen - in der Form von konkreten Sequenzen - entwickelt (Kapitel 4). Dieses Verfahren ermöglicht es, durch die Änderungen sowohl neu hinzugefügtes als auch verloren gegangenes Verhalten des Modells zu berechnen und graphisch zu visualisieren. Mit Hilfe dieser Berechnungen wird das Nachvollziehen von konkreten Änderungen am Modell sowohl leichter als auch transparenter. Insgesamt beantwortet dieses Verfahren Forschungsfrage **RQ2** (*Welche Auswirkungen haben Änderungen an einem Prozessmodell?*) ausführlich.

Deklarative Prozessmodelle gelten als sehr komplex und für den Menschen schwierig oder bisweilen sogar völlig unzugänglich. Deshalb finden diese Modelle trotz ihrer zahlreichen Vorteile gegenüber herkömmlichen imperativen Modellen in der Praxis kaum Beachtung oder Anwendung. Daher wird in Kapitel 6 ein Verfahren entwickelt, das automatenbasiert Lösungen für spezifische Fragestellungen zu konkreten deklarativen Prozessmodellen liefert. Die praktische Nutzbarkeit dieses Verfahrens wird mit Hilfe einer aufwendigen und sorgfältig durchgeführten Benutzerstudie (Kapitel 8.6) demonstriert. Deren Ergebnisse belegen, dass es gelungen ist, Menschen beim Verständnis und bei der Interpretation von deklarativen Prozessmodellen zu unterstützen und dadurch eine Antwort auf Forschungsfrage **RQ3** (*Wie kann die Verständlichkeit deklarativer Prozessmodelle verbessert werden?*) zu liefern.

Alle in dieser Arbeit entwickelten Ansätze wurden in verschiedenen Programmen umgesetzt (Kapitel 7) und mit wissenschaftlicher Sorgfalt und ausführlich evaluiert (Kapitel 8). Dabei wurde das Augenmerk auf sämtliche Aspekte gelegt:

- Die Verfahren werden auf in der Forschung größentechnisch gängigen Beispielprozessmodellen evaluiert.
- Die Verfahren werden auf realen Datensätzen evaluiert.
- Die Verfahren werden bezüglich ihrer Laufzeiten evaluiert.

Man kann zusammengefasst feststellen, dass alle in Kapitel 1.2 formulierten Forschungsfragen umfangreich und zufriedenstellend beantwortet wurden.

9.2 Einschränkungen der entwickelten Ansätze

Obwohl die entwickelten Ansätze bedeutende Fortschritte in der Lösung der adressierten Forschungsfragen aus Kapitel 1.2 darstellen, ist es wichtig, auf einige Einschränkungen hinzuweisen. Diese Einschränkungen zeigen die Grenzen der vorgeschlagenen Methoden auf und ermöglichen eine realistische Einschätzung ihrer Anwendbarkeit und Interpretation der Ergebnisse.

- Die entwickelten Ansätze sind bisher auf die beiden Sprachen BPMN und Declare beschränkt. Diese sind die am weitesten verbreiteten Repräsentanten der imperativen bzw. deklarativen Prozessmodellierung. Nichtsdestotrotz gibt es eine Vielzahl weiterer Sprachen, z.B. DPIL [5] oder DCR Graphs [42], auf welche die entwickelten Ansätze nicht direkt anwendbar sind. Allerdings sind die entwickelten Ansätze und Methoden auf alle Sprachen anwendbar, die in deterministische endliche Automaten transformiert werden können und somit wäre lediglich jeweils eine derartige Transformation anzugeben.

- Für deklarative Prozessmodelle werden in dieser Arbeit die im Standard definierten Declare-Templates (Tabelle 2.2) verwendet. In realen Anwendungen sind diese aber oftmals nicht ausreichend. Deshalb müssen weitere Templates definiert oder gar komplexe Zusammenhänge direkt in LTL formuliert werden. In diesem Fall müssten die korrespondierenden Constraint-Automaten noch berechnet werden. Eine solche Berechnung wäre aber relativ leicht in die entwickelten Ansätze und somit auch in die Implementierung (Kapitel 7.2) integrierbar.

- Für deklarative Prozessmodelle kann das Berechnen des korrespondierenden Prozessautomaten (Kapitel 3.2.2) aufgrund der exponentiellen Zustandsexplosion extrem lange dauern. Die entwickelten Ansätze können somit auf riesige Prozessmodelle, die aus mehreren tausend Constraints bestehen, nicht angewandt werden. Da derart große Prozessmodelle aber realitätsfremd sind, sind die vorgestellten Methoden dennoch praktikabel.

- Ebenso kann die komplette Menge aller Sequenzen bis zu einer bestimmten Länge unter Umständen nicht berechnet werden. Diese Tatsache wird bedingt durch die Anzahl der im Prozessmodell auftretenden Aktivitäten, da die Anzahl der Sequenzen exponentiell von der Anzahl der Aktivitäten abhängt. Dennoch ist es möglich, alle Sequenzen bis zu einer im Prozessmanagement gängigen Sequenzlänge problemlos zu berechnen, was in mehreren Evaluationen in Kapitel 8 demonstriert wird.

- Die definierten Ähnlichkeitsmaße (Kapitel 5) können vereinzelt falsche Eindrücke von Prozessmodellen liefern. Falls ein Ähnlichkeitsmaß eine geringe Ähnlichkeit für zwei Prozessmodelle liefert, bedeutet dies nicht unbedingt, dass diese wirklich unähnlich sind. Hier wird immer der geschulte Blick auf mehrere verschiedene Maße notwendig sein, um qualitative Aussagen treffen zu können.

- Für den Vergleich zusätzlicher Prozessperspektiven (Kapitel 3.3) wird stets vorausgesetzt, dass bereits ein Abgleich der zugrunde liegenden Strukturen, wie beispielsweise der Organisationsmodelle, erfolgt ist. Diese Voraussetzung basiert auf dem Fachwissen von Domänenexpertinnen und Domänenexperten. Jedoch ist zu beachten, dass dieser Standardisierungsprozess äußerst anspruchsvoll und zeitaufwendig sein kann. Daher erfordert er eine sorgfältige Abwägung der potenziellen Vorteile im Vergleich zu den Ressourcen, die für die Durchführung aufgebracht werden müssen.

- Einige der in dieser Arbeit entwickelten Konzepte und Methoden liefern keine empirisch manifestierten Ergebnisse. So erzeugen beispielsweise die in Kapitel 5 definierten Ähnlichkeitsmaße lediglich „Indizien“ für die Ähnlichkeit verschiedener Prozessmodelle. In solchen Fällen sind stets Einschätzungen und Interpretationen von Fachleuten erforderlich.

Die identifizierten Einschränkungen, darunter die Begrenzung auf spezifische Modellierungssprachen wie BPMN und Declare, sowie die potenzielle Zeitintensität bei der Berechnung von Prozessautomaten, stellen zweifellos Herausforderungen dar. Trotzdem verleiht die Flexibilität, die Ansätze auf verschiedene Sprachen anwendbar zu machen, und die Möglichkeit zur Integration weiterer Templates oder komplexer Zusammenhänge, diesen Einschränkungen eine kontextabhängige Perspektive. Die vorgestellten Ähnlichkeitsmaße bieten zwar nur Indizien und erfordern eine kritische Expertenbewertung, dennoch liefern die entwickelten Methoden bedeutende Einblicke und Erkenntnisse. Insgesamt bekräftigt die Dissertation, dass trotz einzelner Limitationen wertvolle Erkenntnisse gewonnen werden können, die einen substanziellen Beitrag zum Verständnis und zur Optimierung von Prozessmodellen leisten.

9.3 Zukünftige Forschungsarbeiten

Abschließend werden noch potenzielle Forschungsthemen beschrieben, die sich aus den Erkenntnissen dieser Dissertation ergeben und thematisch daran anknüpfen können. Diese Themen bauen teilweise direkt auf den in Kapitel 9.2 beschriebenen Einschränkungen auf.

- Da die entwickelten Ansätze auf den beiden Sprachen BPMN und Declare basieren, liegt natürlich eine Erweiterung auf zusätzliche Prozessmodellierungssprachen auf der Hand.

Hier wären vor allem im Bereich der deklarativen Prozessmodellierung Sprachen wie DCR Graphs [42], HiDec [43], GSM [44, 45] oder DPIL [5] interessant. Da alle Methoden und Ansätze auf Automatentheorie beruhen, wäre es hierfür die Herausforderung, ähnliche Transformationen der Hauptkonstrukte der jeweiligen Sprache in deterministische endliche Automaten zu finden.

- Beim Berechnen größerer Prozessautomaten zu Declare-Prozessmodellen, die aus realen Datensätzen extrahiert wurden (Kapitel 8.3.2), konnte festgestellt werden, dass die Reihenfolge, in der die einzelnen Constraint-Automaten miteinander multipliziert werden, unter Umständen einen gravierenden Einfluss auf die Laufzeit besitzen kann. Auf Grund dieser Beobachtung lässt sich vermuten, dass die Berechnung des Prozessautomaten mit den „strikteren“ Constraints begonnen werden sollte, da diese die Anzahl der möglichen Prozessausführungen deutlich reduzieren und somit zu einem Automaten mit weniger Zuständen führen. Dafür wäre ein allgemeingültiges Ranking, das Templates nach „Striktheit“ einordnet, wünschenswert.

- Eine Anregung für eine weitere potentielle Forschungsarbeit ist, das Verfahren zur Berechnung von Verhaltensänderungen in deklarativen Prozessmodellen (Kapitel 4) mit Endnutzerinnen und Endnutzern zu evaluieren. Dafür wäre eine ähnlich wie in Kapitel 8.6 durchgeführte Benutzerstudie sinnvoll, um mit Hilfe von Eyetracking-Experimenten kognitive Auswirkungen des Ansatzes zu untersuchen. Dabei kann gezielt evaluiert werden, inwiefern vor allem die graphische Präsentation der Verhaltensänderungen bei der Überarbeitung deklarativer Prozessmodelle hilft.

- Außerdem könnte eine zusätzliche Benutzerstudie weitere Einblicke in die erarbeiteten Methoden zum Vergleich der organisatorischen Perspektive (Kapitel 3.3.1), der operationalen Perspektive (Kapitel 3.3.2) sowie der Daten- und Datenflussperspektive (Kapitel 3.3.3) liefern. Dabei könnten einerseits Schranken in der Form von konkreten Werten für die Bestimmung von Ähnlichkeiten erforscht und begründet werden. Andererseits kann der praktische Nutzen der definierten Ähnlichkeitsmaße für die Anwendung durch die Endnutzerin bzw. den Endbenutzer aufgezeigt werden.

Insgesamt lässt sich festhalten, dass diese Arbeit praxistaugliche Konzepte und Methoden für den Vergleich von Prozessmodellen, die in verschiedenen Sprachen modelliert wurden, entwickelt. Diese ermöglichen eine effiziente Berechnung von Gemeinsamkeiten und Unterschieden, um eine spätere Fusion der zugrunde liegenden Prozesse zu erleichtern. Zusätzlich können Verhaltensänderungen an Prozessmodellen berechnet werden, um vorgenommene Anpassungen zu verifizieren und sicherzustellen, dass kein ungewolltes Verhalten auftritt. Die Entwicklung

und Umsetzung von Verhaltensanalysen für deklarative Prozessmodelle erleichtern signifikant den Umgang mit diesem Modellierungsparadigma, wie in einer durchgeführten Benutzerstudie gezeigt wird. Diese verdeutlicht, dass Personen in deklarativer Modellierung sowohl schneller als auch sicherer werden, da sie weniger Fehler machen. Dies trägt dazu bei, die Verwendung von deklarativen Prozessmodellen, die im Allgemeinen als komplex gelten und daher in der Praxis selten angewendet werden, deutlich zu erleichtern und praktikabler zu gestalten.

Anhang A

A.1 Constraint-Automaten

Auf den folgenden Seiten sind die Constraint-Automaten zu allen in dieser Arbeit verwendeten Declare-Constraints (Kapitel 2.1.4) aufgelistet.

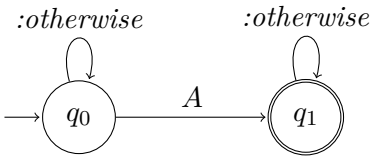


Abbildung A.1: $M_{\text{existence}(A,1)}$

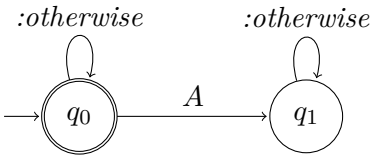


Abbildung A.2: $M_{\text{absence}(A,1)}$

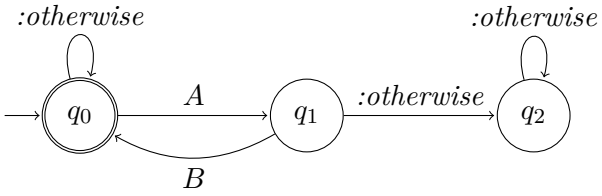


Abbildung A.3: $M_{\text{chainResponse}(A,B)}$

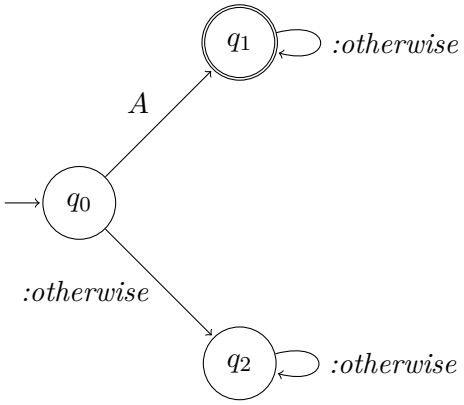


Abbildung A.4: $M_{\text{init}(A)}$

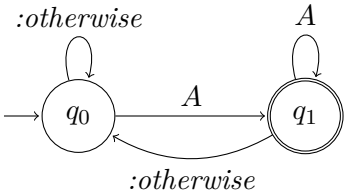


Abbildung A.5: $M_{\text{last}(A)}$

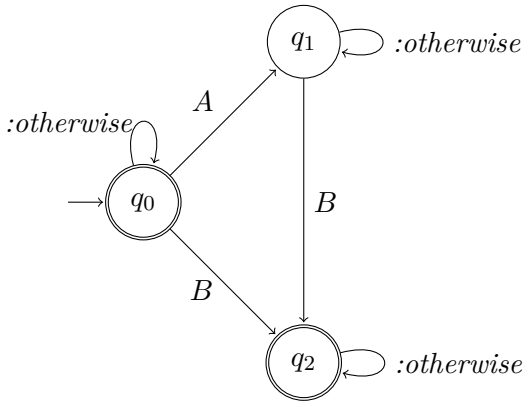


Abbildung A.6: $M_{\text{respondedExistence}(A,B)}$

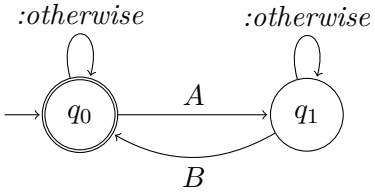


Abbildung A.7: $M_{\text{response}(A,B)}$

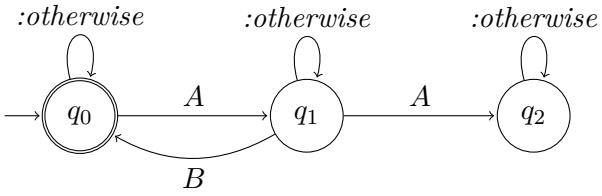


Abbildung A.8: $M_{\text{alternateResponse}(A,B)}$

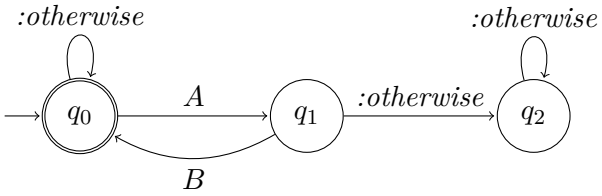


Abbildung A.9: $M_{\text{chainResponse}(A,B)}$

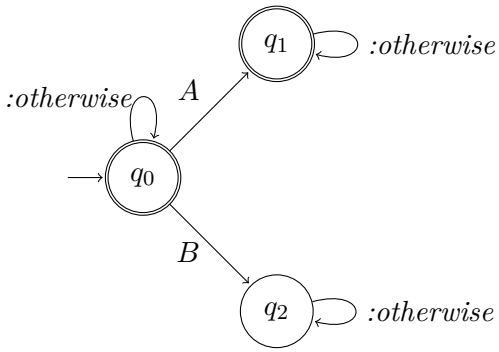


Abbildung A.10: $M_{\text{precedence}(A,B)}$

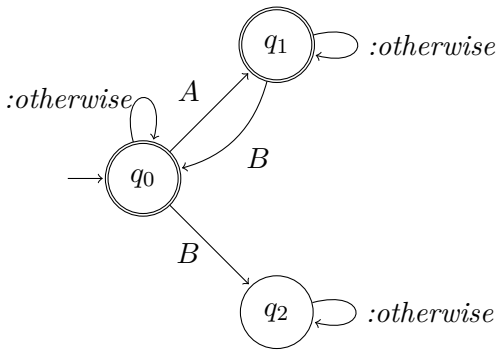


Abbildung A.11: $M_{\text{alternatePrecedence}(A,B)}$

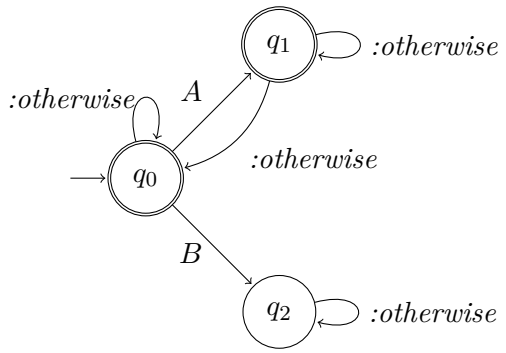


Abbildung A.12: $M_{\text{chainPrecedence}(A,B)}$

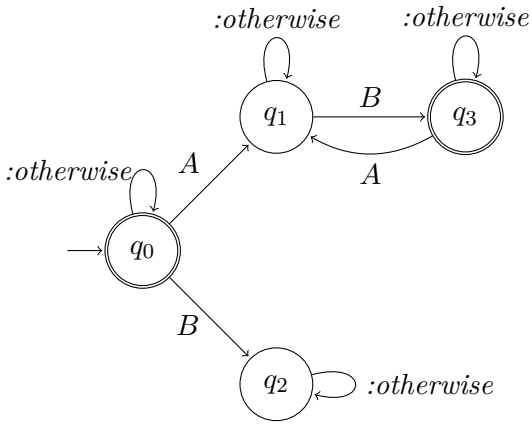


Abbildung A.13: $M_{\text{succession}(A,B)}$

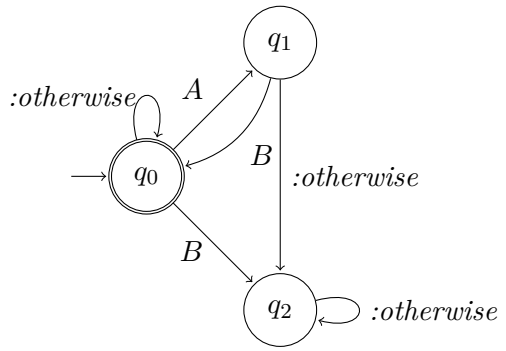


Abbildung A.14: $M_{\text{chainSuccession}(A,B)}$

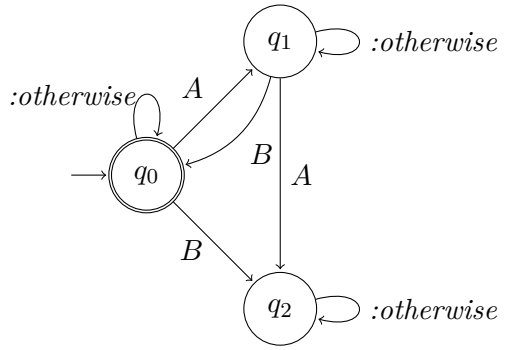


Abbildung A.15: $M_{\text{alternateSuccession}(A,B)}$

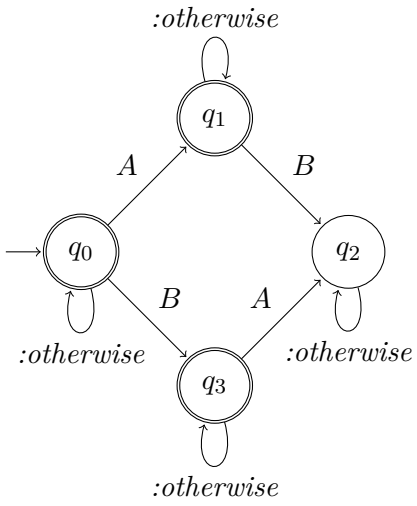


Abbildung A.16: $M_{\text{notRespondedExistence}(A,B)}$

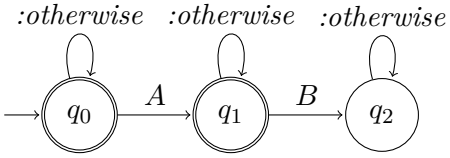


Abbildung A.17: $M_{\text{notResponse}(A,B)}$

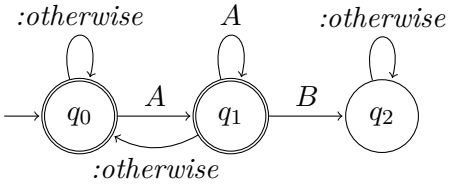


Abbildung A.18: $M_{\text{notChainResponse}(A,B)}$

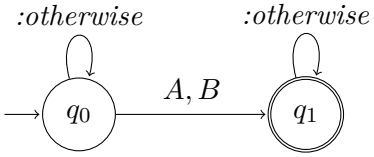


Abbildung A.19: $M_{\text{choice}(A,B)}$

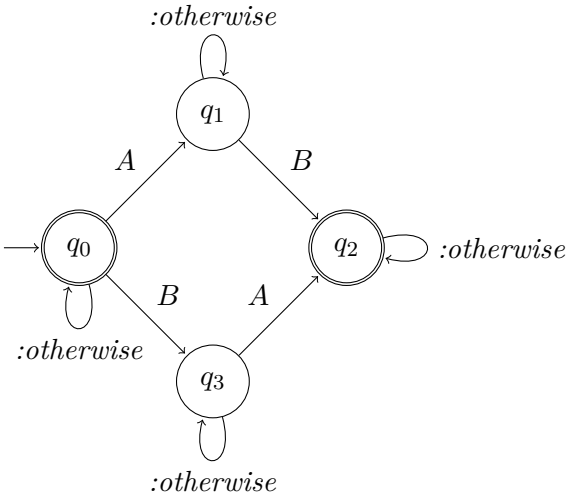


Abbildung A.20: $M_{\text{coExistence}(A,B)}$

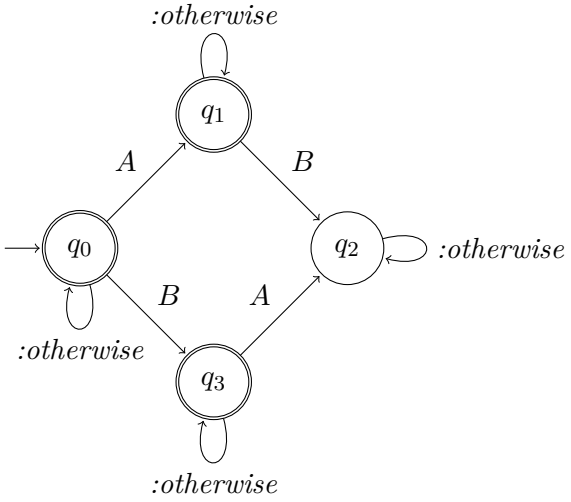


Abbildung A.21: $M_{\text{notCoExistence}(A,B)}$

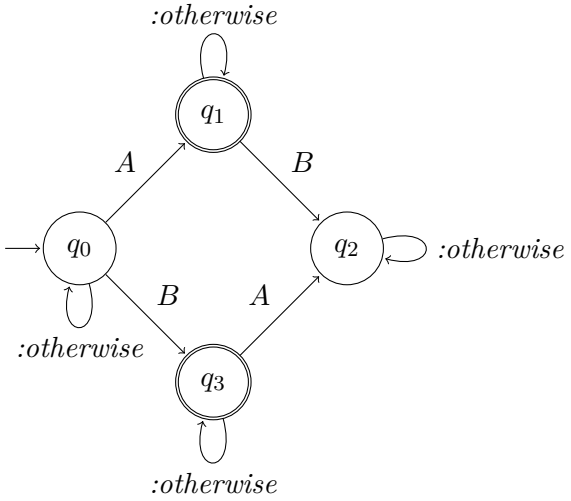


Abbildung A.22: $M_{\text{exclusiveChoice}(A,B)}$

A.2 Automaten für Beispieltransformation

Auf den folgenden Seiten sind alle weiteren zur Transformation des Declare Modells P aus Kapitel 3.2.2 benötigten DEAs zu finden.

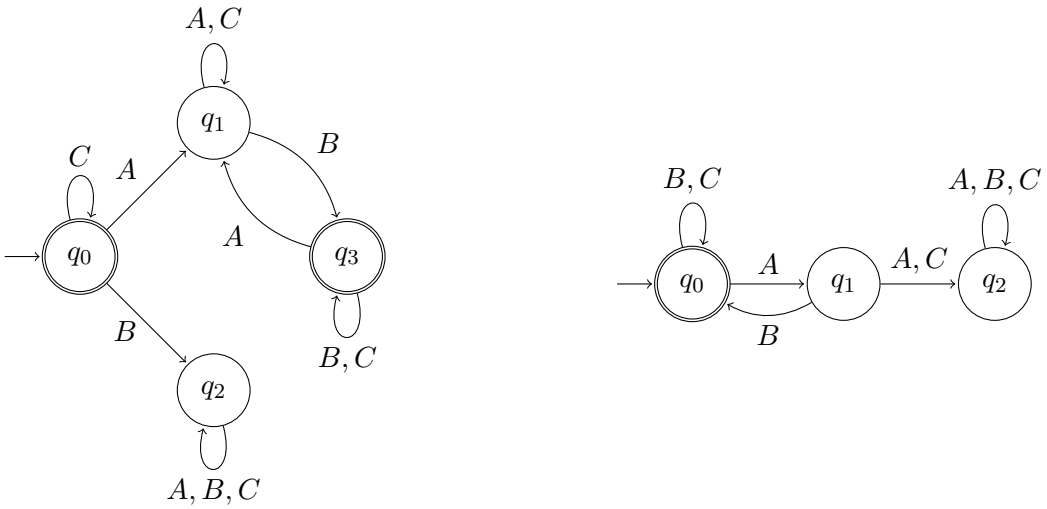


Abbildung A.23: Constraint-Automaten $M_{\text{succession}(A,B)}$ (links) und $M_{\text{chainResponse}(A,B)}$ (rechts)

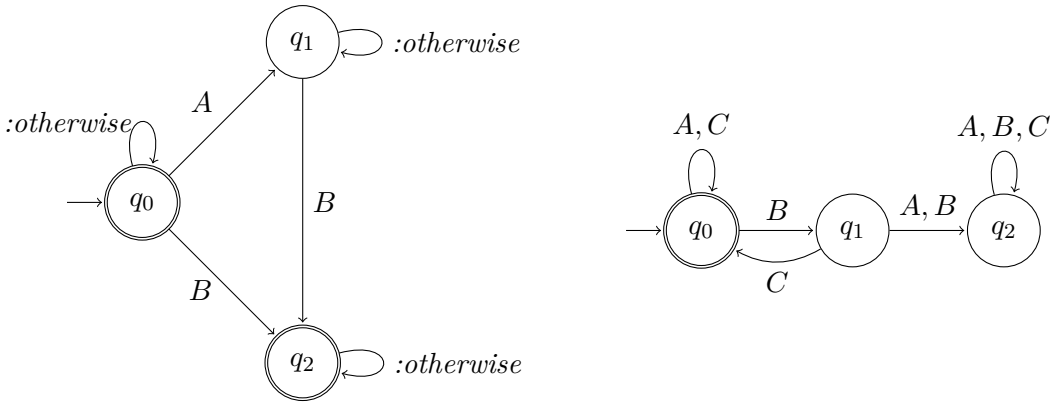


Abbildung A.24: Constraint-Automaten $M_{\text{respondedExistence}(A,B)}$ (links) und $M_{\text{chainResponse}(B,C)}$ (rechts)

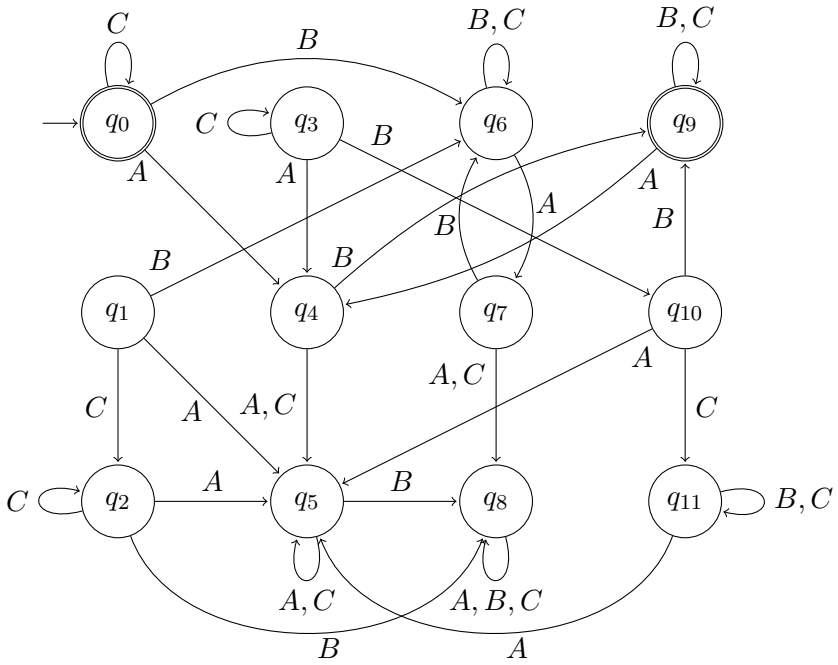


Abbildung A.25: Produktautomat $M_1 := M_{\text{succession}(A,B)} \times M_{\text{chainResponse}(A,B)}$

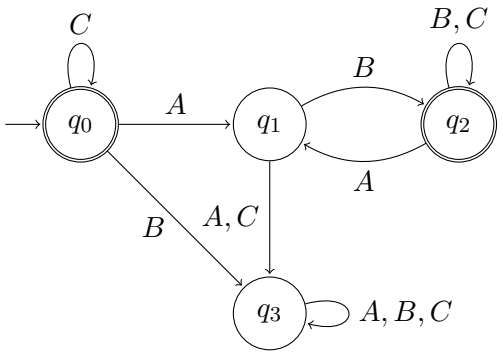


Abbildung A.26: Zu M_1 äquivalenter Minimalautomat $M_{1_{min}}$

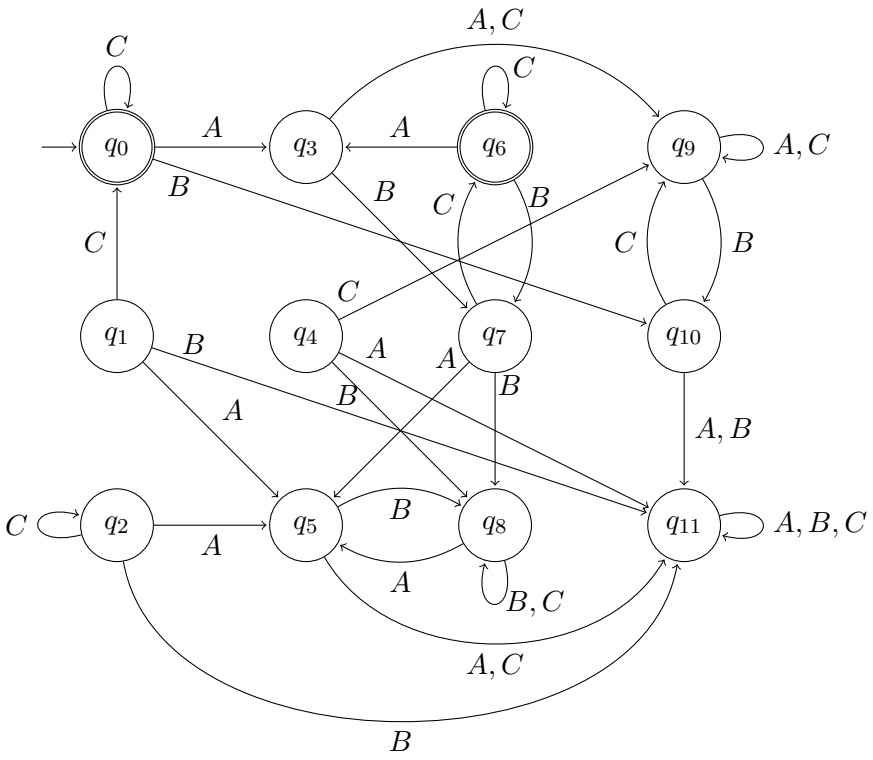


Abbildung A.27: Produktautomat $M_2 := M_{1_{min}} \times M_{\text{chainResponse}(B,C)}$

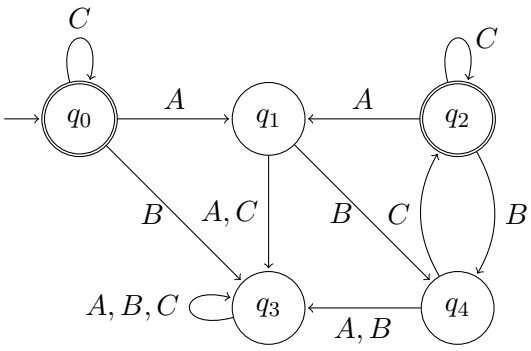


Abbildung A.28: Zu M_2 äquivalenter Minimalautomat $M_{2_{min}}$

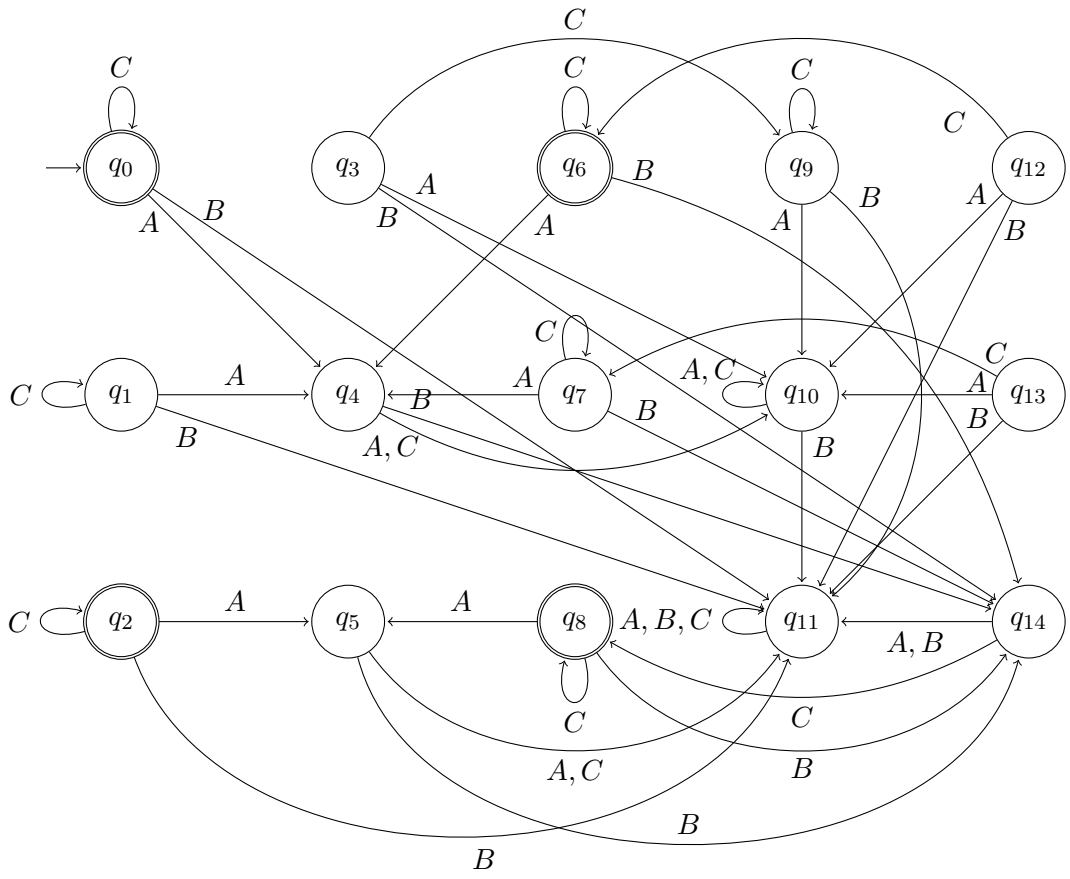


Abbildung A.29: Produktautomat $M_3 := M_{2_{min}} \times M_{\text{respondedExistence}(A,B)}$

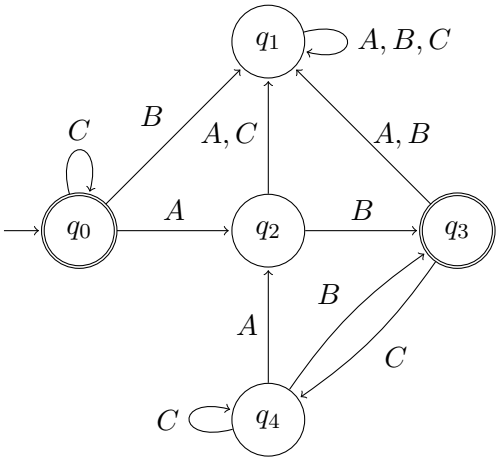


Abbildung A.30: Zu M_3 äquivalenter Minimalautomat $M_{3_{min}}$ (entspricht dem Prozessautomaten)

A.3 Zusatzmaterialien zur Benutzerstudie

In diesem Abschnitt sind die beiden Seiten des Datenblattes zu finden, das für die Erfassung von Daten der Studienteilnehmenden verwendet wurde.

Alter			
Geschlecht		☐ m ☐ w ☐ d	
Höchster akademischer Abschluss		☐ keiner ☐ Real / Mittelschule ☐ Abitur / Fachabitur ☐ Bachelor ☐ Master ☐ Promotion ☐_____	
Fokus der Ausbildung			
Derzeit ausgeübter Beruf			
Relevantes Feld			
Berufserfahrung im relevanten Feld		____ Jahre	
Vorkenntnisse über Prozessmodelle (UML Aktivitätsdiagramm, Flussdiagramm, BPMN Modell)		☐ Ja ☐ Nein	
Nur falls vorherige Frage mit ja beantwortet wurde	Prozessmodelle allgemein (lesen und erstellen):	Lesen ☐ ja ☐ nein Erstellen ☐ ja ☐ nein	
	Deklarative Prozessmodelle	Lesen ☐ ja ☐ nein Erstellen ☐ ja ☐ nein	
	Falls vorherige Frage(n) mit „ja“ beantwortet wurde(n): Erfahrung in Jahren	Prozessmodelle allgemein: _____ Jahre Deklarative Prozessmodelle: _____ Jahre	
Bearbeitungsreihenfolge		☐ zunächst ohne Tool ☐ zunächst mit Tool	
Beginn mit Aufgabenblock		☐ 1 ☐ 2	
Reihenfolge der Aufgaben		**AUFGABENBLOCK 1** ☐ 123 ☐ 132 ☐ 213 ☐ 231 ☐ 312 ☐ 321	**AUFGABENBLOCK 2** ☐ 123 ☐ 132 ☐ 213 ☐ 231 ☐ 312 ☐ 321
Bearbeitungszeiten & Bewertungen (0: falsch, 1: teilweise richtig, 2: richtig, #: abgebrochen)		**AUFGABENBLOCK 1** 1a) ____ Sek.; /5 2a) ____ Sek.; 0/1/2/# 2b) ____ Sek.; 0/2/# 3a) ____ Sek.; /8 3b) ____ Sek.; /4 3c) ____ Sek.; 0/1/2/#	**AUFGABENBLOCK 2** 1a) ____ Sek.; /4 2a) ____ Sek.; 0/1/2/# 2b) ____ Sek.; 0/2/# 3a) ____ Sek.; /7 3b) ____ Sek.; /6 3c) ____ Sek.; 0/1/2/#
Fehlerbeschreibung			

Abbildung A.31: Datenblatt Seite 1

Angabenzettel wurde bei Bearbeitung mit Tool verwendet	☐ Ja ☐ Nein	

Bitte bewerten Sie Ihre geistige Anstrengung in Bezug auf die gerade absolvierte Folge von Aufgaben durch ein einzelnes Kreuz auf der folgenden Skala.

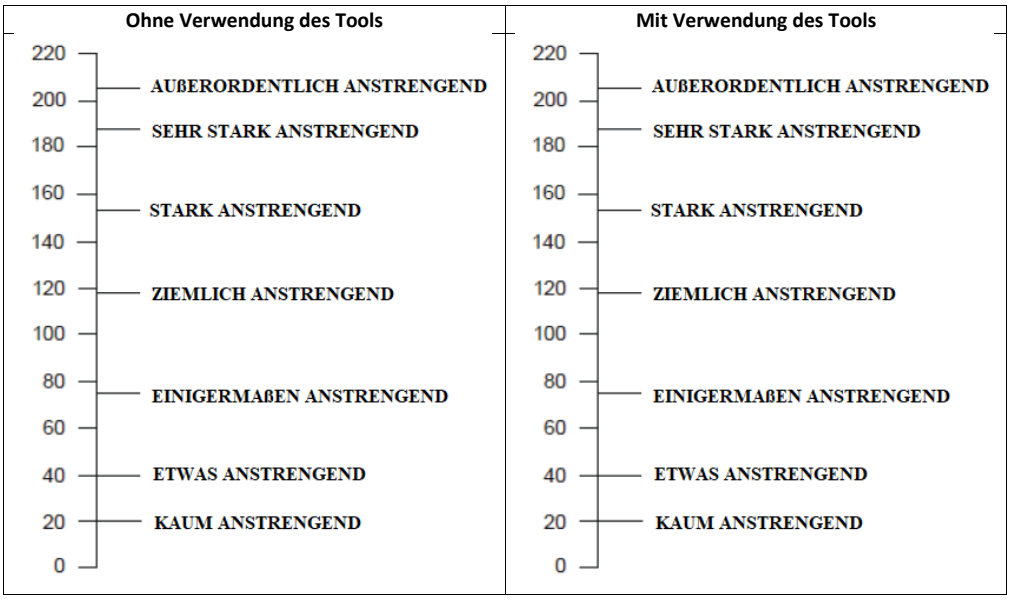


Abbildung A.32: Datenblatt Seite 2

Anhang B

Vollständige Liste eigener Publikationen

[24] *Execution of Multi-perspective Declarative Process Models*
L. Ackermann, S. Schönig, S. Petter, N. Schützenmeier, S. Jablonski, 26th International Conference on Cooperative Information Systems, Springer, Malta, 2018.

[25] *Logic Based Look-Ahead for the Execution of Multi-perspective Declarative Processes*
M. Käppel, N. Schützenmeier, S. Schönig, L. Ackermann, S. Jablonski, 20th International Conference on Business Process Modeling, Development and Support, Springer, Rom, 2019.

[26] *Detection of Declarative Process Constraints in LTL Formulas*
N. Schützenmeier, M. Käppel, S. Petter, S. Schönig, S. Jablonski, 15th International Workshop on Enterprise and Organizational Modeling and Simulation, Springer, Rom, 2019.

[27] *A Comparatative Study for the Selection of Machine Learning Algorithms based on Decriptive Parameters*
C. Kumar, M. Käppel, N. Schützenmeier, P. Eisenhuth, S. Jablonski, 8th International Conference on Data Science, Technology and Applications, SciTePress, Prag, 2019.

[8] *Towards a Hybrid Process Modeling Language*
N. Schützenmeier, S. Jablonski, S. Schönig, 15th International Conference on Research Challenges in Information Science, Springer, Limassol, 2021.

[21] *Upper-Bounded Model Checking for Declarative Process Models*
N. Schützenmeier, M. Käppel, S. Petter, S. Jablonski, 14th Working Conference on the Practice of Enterprise Modeling, Springer, Riga, 2021.

[22] *Automaton-based comparison of Declare process models*
N. Schützenmeier, M. Käppel, L. Ackermann, S. Jablonski, S. Petter, Software and Systems Modeling 22, 2022.

[18] *Scenario-Based Model Checking of Declarative Process Models*
N. Schützenmeier, M. Käppel, M. Fichtner, S. Jablonski, 25th International Conference on Enterprise Information Systems, SciTePress, Prag, 2023.

[23] *Efficient Computation of Behavioral Changes in Declarative Process Models*
N. Schützenmeier, C. Corea, P. Delfmann, S. Jablonski, 24th International Conference on Business Process Modeling, Development and Support, Springer, Saragossa, 2023.

[11] *Comparing the Expressiveness of Imperative and Declarative Process Models*
N. Schützenmeier, S. Jablonski, M. Käppel, L. Ackermann, 3rd International Workshop on Model-Driven Organizational and Business Agility, Springer, Saragossa, 2023.

Literatur

[1] Wil M. P. van der Aalst. *Process Mining: Data Science in Action*. 2. 2nd ed. 2016. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016. ISBN: 9783662498514.

[2] Jan vom Brocke und Michael Rosemann. *Strategic alignment, governance, people and culture*. Bd. 2. International handbooks on information systems. Heidelberg: Springer, 2010. ISBN: 978-3-642-01981-4. DOI: 10.1007/978-3-642-01982-1.

[3] Dirk Fahland, Daniel Lübke, Jan Mendling, Hajo Reijers, Barbara Weber, Matthias Weidlich und Stefan Zugal. „Declarative versus Imperative Process Modeling Languages: The Issue of Understandability". In: *Enterprise, business process and information systems modeling*. Hrsg. von Terry Halpin. Bd. 29. Lecture Notes in Business Information Processing. Berlin und Heidelberg: Springer, 2009, S. 353–366. ISBN: 978-3-642-01861-9. DOI: 10.1007/978-3-642-01862-6_29.

[4] Dirk Fahland, Jan Mendling, Hajo A. Reijers, Barbara Weber, Matthias Weidlich und Stefan Zugal. „Declarative versus Imperative Process Modeling Languages: The Issue of Maintainability". In: *Business Process Management Workshops*. Hrsg. von Stefanie Rinderle-Ma. Bd. 43. Lecture Notes in Business Information Processing. Berlin und Heidelberg: Springer, 2010, S. 477–488. ISBN: 978-3-642-12185-2. DOI: 10.1007/978-3-642-12186-9_45.

[5] Stefan Schönig und Michael Zeising. „The DPIL Framework: Tool Support for Agile and Resource-Aware Business Processes". In: Bd. 1418. 2015.

[6] Maja Pesic, Helen Schonenberg und Wil M.P. van der Aalst. „DECLARE: Full Support for Loosely-Structured Processes". In: *Proceedings / 11th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2007*. Los Alamitos, Calif.: IEEE Computer Society, 2007, S. 287. ISBN: 0-7695-2891-0. DOI: 10.1109/EDOC.2007.14.

[7] José Luís Pereira und Diogo Silva. „Business Process Modeling Languages: A Comparative Framework". In: *New Advances in Information Systems and Technologies*. Hrsg. von Álvaro Rocha, Ana Maria Correia, Hojjat Adeli, Luis Paulo Reis und Marcelo Mendonça

Teixeira. Bd. 444. Advances in Intelligent Systems and Computing. Cham und s.l.: Springer International Publishing, 2016, S. 619–628. ISBN: 978-3-319-31231-6. DOI: 10.1007/978-3-319-31232-3_58.

[8] Nicolai Schützenmeier, Stefan Jablonski und Stefan Schönig. „Towards a Hybrid Process Modeling Language“. In: *Research Challenges in Information Science*. Hrsg. von Samira Cherfi, Anna Perini und Selmin Nurcan. Bd. 415. Springer eBook Collection. Cham: Springer International Publishing und Imprint Springer, 2021, S. 630–636. ISBN: 978-3-030-75017-6. DOI: 10.1007/978-3-030-75018-3_46.

[9] M. Pesic. *Constraint-based workflow management systems:shifting control to users*. 2008. DOI: 10.6100/IR638413.

[10] Jan Mendling, Marlon Dumas, Marcello La Rosa und Hajo A. Reijers. *Grundlagen des Geschäftsprozessmanagements: Übersetzt von Thomas Grisold, Steven Groß, Jan Mendling, Bastian Wurm*. 1. Auflage. Berlin: Springer Vieweg, 2021. ISBN: 9783662587362.

[11] Nicolai Schützenmeier, Stefan Jablonski, Martin Käppel und Lars Ackermann. „Comparing the Expressiveness of Imperative and Declarative Process Models“. In: *Model-Driven Organizational and Business Agility*. Hrsg. von Eduard Babkin, Joseph Barjis, Pavel Malyzhenkov, Vojtěch Merunka und Martin Molhanec. Bd. 488. Lecture Notes in Business Information Processing. [S.l.]: Springer International, 2023, S. 16–31. ISBN: 978-3-031-45009-9. DOI: 10.1007/978-3-031-45010-5_2.

[12] Chen Li, Manfred Reichert und Andreas Wombacher. „On Measuring Process Model Similarity Based on High-Level Change Operations“. In: *Conceptual modeling - ER 2008*. Hrsg. von Qing Li, Stefano Spaccapietra, Eric Yu und Antoni Olivé. Bd. 5231. Lecture Notes in Computer Science. Berlin und Heidelberg: Springer, 2008, S. 248–264. ISBN: 978-3-540-87876-6. DOI: 10.1007/978-3-540-87877-3_19.

[13] Bill Curtis, Marc Kellner und Jim Over. „Process Modeling“. In: *Commun. ACM* 35 (1992), S. 75–90. DOI: 10.1145/130994.130998.

[14] Stefan Jablonski und Christoph Bussler. *Workflow management: Modeling concepts, architecture and implementation*. London: ITP Internat. Thomson Computer Press, 1996. ISBN: 1850322228.

[15] Claudio Di Ciccio, Fabrizio Maria Maggi, Marco Montali und Jan Mendling. „Resolving inconsistencies and redundancies in declarative process models“. In: *Information Systems* 64 (2017), S. 425–446. ISSN: 03064379. DOI: 10.1016/j.is.2016.09.005.

[16] Carl Corea und Patrick Delfmann. „Quasi-Inconsistency in Declarative Process Models“. In: *Business Process Management Forum*. Hrsg. von Thomas Hildebrandt, Boudewijn F. van Dongen, Maximilian Röglinger und Jan Mendling. Bd. 360. Springer eBooks Computer Science. Cham: Springer, 2019, S. 20–35. ISBN: 978-3-030-26642-4. DOI: 10.1007/978-3-030-26643-1_2.

[17] Cornelia Haisjackl, Irene Barba, Stefan Zugal, Pnina Soffer, Irit Hadar, Manfred Reichert, Jakob Pinggera und Barbara Weber. „Understanding Declare models: strategies, pitfalls, empirical results“. In: *Software and Systems Modeling* 15.2 (2016), S. 325–352. ISSN: 1619-1374. DOI: 10.1007/s10270-014-0435-z.

[18] Nicolai Schützenmeier, Martin Käppel, Myriel Fichtner und Stefan Jablonski. „Scenario-Based Model Checking of Declarative Process Models“. In: *Proceedings of the 25th International Conference on Enterprise Information Systems*. SCITEPRESS - Science and Technology Publications, 2023, S. 406–417. ISBN: 978-989-758-648-4. DOI: 10.5220/0011856400003467.

[19] Johannes de Smedt, Jochen de Weerdt, Estefanía Serral und Jan Vanthienen. „Improving Understandability of Declarative Process Models by Revealing Hidden Dependencies“. In: *Advanced information systems engineering*. Hrsg. von Selmin Nurcan, Pnina Soffer, Marko Bajec und Johann Eder. Bd. 9694. Lecture notes in computer science Information systems and applications, incl. Internet/web, and HCI. Cham: Springer, 2016, S. 83–98. ISBN: 978-3-319-39695-8. DOI: 10.1007/978-3-319-39696-5_6.

[20] Hevner, March, Park und Ram. „Design Science in Information Systems Research“. In: *MIS Quarterly* 28.1 (2004), S. 75. ISSN: 02767783. DOI: 10.2307/25148625.

[21] Nicolai Schützenmeier, Martin Käppel, Sebastian Petter und Stefan Jablonski. „Upper-Bounded Model Checking for Declarative Process Models“. In: *Practice of Enterprise Modeling : 14th IFIP WG 8. 1 Working Conference, PoEM 2021, Riga, Latvia, November 24-26, 2021, Proceedings*. Hrsg. von Estefanía Serral. Bd. 432. Lecture Notes in Business Information Processing. Cham: Springer International Publishing AG, 2021, S. 195–211. ISBN: 978-3-030-91278-9. DOI: 10.1007/978-3-030-91279-6_14.

[22] Nicolai Schützenmeier, Martin Käppel, Lars Ackermann, Stefan Jablonski und Sebastian Petter. „Automaton-based comparison of Declare process models“. In: *Software and Systems Modeling* (2022). ISSN: 1619-1374. DOI: 10.1007/s10270-022-01069-y.

[23] Nicolai Schützenmeier, Carl Corea, Patrick Delfmann und Stefan Jablonski. „Efficient Computation of Behavioral Changes in Declarative Process Models“. In: *Enterprise, Business-Process and Information Systems Modeling*. Hrsg. von Han van der Aa, Dominik Bork, Henderik A. Proper und Rainer Schmidt. Bd. 479. Lecture Notes in Business

Information Processing. Cham: Springer Nature Switzerland und Imprint Springer, 2023, S. 136–151. ISBN: 978-3-031-34240-0. DOI: 10.1007/978-3-031-34241-7_10.

[24] Lars Ackermann, Stefan Schönig, Sebastian Petter, Nicolai Schützenmeier und Stefan Jablonski. „Execution of Multi-perspective Declarative Process Models“. In: *On the move to meaningful internet systems*. Hrsg. von Hervé Panetto, Christophe Debruyne, Erik Proper, Claudio Agostino Ardagna, Dumitu Roman und Robert Meersman. Bd. 11230. Lecture Notes in Computer Science Programming and software engineering. Cham: Springer, 2018, S. 154–172. ISBN: 978-3-030-02670-7. DOI: 10.1007/978-3-030-02671-4_9.

[25] Martin Käppel, Nicolai Schützenmeier, Stefan Schönig, Lars Ackermann und Stefan Jablonski. „Logic Based Look-Ahead for the Execution of Multi-perspective Declarative Processes“. In: *Enterprise, Business-Process and Information Systems Modeling*. Hrsg. von Iris Reinhartz-Berger, Jelena Zdravkovic, Jens Gulden und Rainer Schmidt. Bd. 352. Springer eBooks Computer Science. Cham: Springer, 2019, S. 53–68. ISBN: 978-3-030-20617-8. DOI: 10.1007/978-3-030-20618-5_4.

[26] Nicolai Schützenmeier, Martin Käppel, Sebastian Petter, Stefan Schönig und Stefan Jablonski. „Detection of Declarative Process Constraints in LTL Formulas“. In: *Enterprise and Organizational Modeling and Simulation*. Hrsg. von Robert Pergl, Eduard Babkin, Russell Lock, Pavel Malyzhenkov und Vojtěch Merunka. Bd. 366. Springer eBooks Computer Science. Cham: Springer, 2019, S. 131–145. ISBN: 978-3-030-35645-3. DOI: 10.1007/978-3-030-35646-0_10.

[27] Chettan Kumar, Martin Käppel, Nicolai Schützenmeier, Philipp Eisenhuth und Stefan Jablonski. „A Comparative Study for the Selection of Machine Learning Algorithms based on Descriptive Parameters“. In: *DATA 2019*. Hrsg. von Slimane Hammoudi, Christoph Quix und Jorge Bernardino. Setúbal, Portugal: SCITEPRESS - Science and Technology Publications Lda, 2019, S. 408–415. ISBN: 978-989-758-377-3. DOI: 10.5220/0008117404080415.

[28] H. James Harrington. *Business process improvement: The breakthrough strategy for total quality, productivity, and competitiveness*. 23. [Nachdr.] Boston, Mass.: McGraw-Hill, 2007. ISBN: 9780070267688.

[29] *Business process management: BPM common body of knowledge - BPM CBOK ; Leitfaden für das Prozessmanagement*. Version 2.0. Bd. Bd. 1. Schriftenreihe der EABPM. Gießen: G.Schmidt, 2009. ISBN: 9783921313800.

[30] Marlon Dumas. *Fundamentals of Business Process Management*. SpringerLink Bücher. Berlin und Heidelberg: Springer, 2013. ISBN: 9783642331435. DOI: 10.1007/978-3-642-33143-5.

[31] Stefan Schönig, Michael Zeising und Stefan Jablonski. „Towards Location-Aware Declarative Business Process Management“. In: *Business Information Systems Workshops*. Hrsg. von Witold Abramowicz und Angelika Kokkinaki. Bd. 183. Lecture Notes in Business Information Processing. Cham, Heidelberg und New York: Springer, 2014, S. 40–51. ISBN: 978-3-319-11459-0. DOI: 10.1007/978-3-319-11460-6_4.

[32] M. Baumann und S. Jablonski. *Über die Anwendung, Klassifizierung und Übertragbarkeit von Methoden für einen Ähnlichkeitsabgleich von Geschäftsprozessmodellen*. Universität Bayreuth, 2018.

[33] Michael Schlundt. *Historienverwaltung in Workflow-Management-Systemen: Zugl.: Erlangen, Nürnberg, Univ., Diss., 2004*. Berichte aus der Informatik. Aachen: Shaker, 2004. ISBN: 3832235558.

[34] August-Wilhelm Scheer, Oliver Thomas und Otmar Adam. „Process Modeling using Event-Driven Process Chains“. In: *Process-aware information systems*. Hrsg. von Marlon Dumas. Hoboken, NJ: Wiley, 2005, S. 119–145. ISBN: 9780471741442. DOI: 10.1002/0471741442.ch6.

[35] W.M.P. van der Aalst und A.H.M. ter Hofstede. „YAWL: yet another workflow language“. In: *Information Systems* 30.4 (2005), S. 245–275. ISSN: 03064379. DOI: 10.1016/j.is.2004.02.002.

[36] Jan Recker. „Opportunities and constraints: the current struggle with BPMN“. In: *Business Process Management Journal* 16.1 (2010), S. 181–201. ISSN: 1463-7154. DOI: 10.1108/14637151011018001.

[37] P. Wohed, W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede und N. Russell. „On the Suitability of BPMN for Business Process Modelling“. In: *Business process management*. Hrsg. von Schahram Dustdar. Bd. 4102. Lecture Notes in Computer Science. Berlin und Heidelberg: Springer, 2006, S. 161–176. ISBN: 978-3-540-38901-9. DOI: 10.1007/11841760_12.

[38] Bendick Mahleko und Andreas Wombacher. „Indexing Business Processes based on Annotated Finite State Automata“. In: *2006 IEEE International Conference on Web Services (ICWS'06)*. 2006, S. 303–311. DOI: 10.1109/ICWS.2006.74.

[39] W. M. P. van der Aalst, M. Pesic und H. Schonenberg. „Declarative workflows: Balancing between flexibility and support“. In: *Computer Science - Research and Development* 23.2 (2009), S. 99–113. ISSN: 1865-2034. DOI: 10.1007/s00450-009-0057-9.

[40] Stefan Schönig, Lars Ackermann und Stefan Jablonski. „Towards an Implementation of Data and Resource Patterns in Constraint-based Process Models“. In: *MODELSWARD 2018*. Hrsg. von Slimane Hammoudi, Luis Ferreira Pires und Bran Selic. Setúbal, Portugal: SCITEPRESS - Science and Technology Publications Lda, 2018, S. 271–278. ISBN: 978-989-758-283-7. DOI: 10.5220/0006533502710278.

[41] Michael Zeising, Stefan Schönig und Stefan Jablonski. „Towards a Common Platform for the Support of Routine and Agile Business Processes“. In: *2014 International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2014)*. Hrsg. von Elisa Bertino, Shu-Ching Chen, Karl Aberer, Prashant Krishnamurthy und Murat Kantarcioglu. Piscataway, NJ: IEEE, 2014. ISBN: 978-1-63190-043-3. DOI: 10.4108/icst.collaboratecom.2014.257269.

[42] Thomas Hildebrandt, Raghava Rao Mukkamala, Tijs Slaats und Francesco Zanitti. „Contracts for cross-organizational workflows as timed Dynamic Condition Response Graphs“. In: *The Journal of Logic and Algebraic Programming* 82.5-7 (2013), S. 164–185. ISSN: 15678326. DOI: 10.1016/j.jlap.2013.05.005.

[43] Riccardo de Masellis, Chiara Di Francescomarino, Chiara Ghidini und Fabrizio M. Maggi. „Declarative Process Models: Different Ways to Be Hierarchical“. In: *Service-oriented computing*. Hrsg. von Quan Z. Sheng, Eleni Stroulia, Samir Tata und Sami Bhiri. Bd. 9936. Lecture Notes in Computer Science. Cham: Springer, 2016, S. 104–119. ISBN: 978-3-319-46294-3. DOI: 10.1007/978-3-319-46295-0_7.

[44] Richard Hull, Elio Damaggio, Fabiana Fournier, Manmohan Gupta, Fenno Heath, Stacy Hobson, Mark Linehan, Sridhar Maradugu, Anil Nigam, Piyawadee Sukaviriya und Roman Vaculin. „Introducing the Guard-Stage-Milestone Approach for Specifying Business Entity Lifecycles“. In: *Web services and formal methods*. Hrsg. von Mario Bravetti und Tevfik Bultan. Bd. 6551. Lecture Notes in Computer Science / Programming and Software Engineering. Berlin und Heidelberg: Springer, 2011, S. 1–24. ISBN: 978-3-642-19588-4. DOI: 10.1007/978-3-642-19589-1_1.

[45] Richard Hull, Elio Damaggio, Riccardo de Masellis, Fabiana Fournier, Manmohan Gupta, Fenno Terry Heath, Stacy Hobson, Mark Linehan, Sridhar Maradugu, Anil Nigam, Piwadee Noi Sukaviriya und Roman Vaculin. „Business artifacts with guard-stage-milestone lifecycles“. In: *Proceedings of the 5th ACM international conference on Distributed event-based system*. Hrsg. von Opher Etzion. ACM Conferences. New York, NY: ACM, 2011, S. 51–62. ISBN: 9781450304238. DOI: 10.1145/2002259.2002270.

[46] Andrea Burattin, Fabrizio M. Maggi und Alessandro Sperduti. „Conformance checking based on multi-perspective declarative process models“. In: *Expert Systems with Applications* 65 (2016), S. 194–211. ISSN: 09574174. DOI: 10.1016/j.eswa.2016.08.040.

[47] Michael Huth und Mark Ryan. *Logic in computer science: Modelling and reasoning about systems*. 4.ed., 8.print. Cambridge: Cambridge University Press, 2012. ISBN: 9780521543101.

[48] Christel Baier und Joost-Pieter Katoen. *Principles of model checking*. The MIT Press Ser. Cambridge, Massachusetts und London, England: The MIT Press, 2008. ISBN: 9780262267564.

[49] Fabrizio M. Maggi, R. P. Jagadeesh Chandra Bose und Wil M. P. van der Aalst. „Efficient Discovery of Understandable Declarative Process Models from Event Logs“. In: *Active Flow and Combustion Control 2018*. Hrsg. von Rudibert King. Bd. 141. Notes on Numerical Fluid Mechanics and Multidisciplinary Design Ser. Cham: Springer International Publishing AG, 2019, S. 270–285. ISBN: 978-3-319-98176-5. DOI: 10.1007/978-3-642-31095-9_18.

[50] John E. Hopcroft, Rajeev Motwani und Jeffrey David Ullman. *Introduction to automata theory, languages, and computation*. 3rd ed. Boston: Pearson Addison-Wesley, 2007. ISBN: 0321455363.

[51] John Hopcroft. „AN n log n ALGORITHM FOR MINIMIZING STATES IN A FINITE AUTOMATON“. In: *Theory of machines and computations;* hrsg. von Zvi Kohavi und Azaria Paz. New York: Academic Press, 1971, S. 189–196. ISBN: 9780124177505. DOI: 10.1016/B978-0-12-417750-5.50022-1.

[52] Josep Sànchez-Ferreres, Han van der Aa, Josep Carmona und Lluís Padró. „Aligning textual and model-based process descriptions“. In: *Data & Knowledge Engineering* 118 (2018), S. 25–40. ISSN: 0169023X. DOI: 10.1016/j.datak.2018.09.001.

[53] Remco Dijkman, Marlon Dumas, Luciano Garcia-Banuelos und Reina Kaarik. „Aligning Business Process Models“. In: *2009 13th IEEE International Enterprise Distributed Object Computing Conference*. Piscataway, NJ: IEEE, 2009, S. 45–53. ISBN: 978-0-7695-3785-6. DOI: 10.1109/EDOC.2009.11.

[54] Christopher Klinkmüller, Ingo Weber, Jan Mendling, Henrik Leopold und André Ludwig. „Increasing Recall of Process Model Matching by Improved Activity Label Matching“. In: *Business process management*. Hrsg. von Florian Daniel, Jianmin Wang und Barbara Weber. Bd. 8094. Lecture Notes in Computer Science. Berlin: Springer, 2013, S. 211–218. ISBN: 978-3-642-40175-6. DOI: 10.1007/978-3-642-40176-3_17.

[55] M. Westergaard, C. Stahl und H. A. Reijers. *UnconstrainedMiner : efficient discovery of generalized declarative process models*. BPM reports. BPMcenter. org, 2013.

[56] Michael Heinrich Baumann, Michaela Baumann, Stefan Schönig und Stefan Jablonski. „Towards Multi-perspective Process Model Similarity Matching“. In: *Enterprise and organizational modeling and simulation*. Hrsg. von Joseph Barjis. Bd. 191. Lecture Notes in Business Information Processing. Heidelberg: Springer, 2014, S. 21–37. ISBN: 978-3-662-44859-5. DOI: 10.1007/978-3-662-44860-1_2.

[57] Michaela Baumann, Michael Heinrich Baumann, Stefan Schönig und Stefan Jablonski. „Resource-Aware Process Model Similarity Matching“. In: *Service-oriented computing - ICSOC 2014 workshops*. Hrsg. von Farouk Toumani, Barbara Pernici, Daniela Grigori, Djamal Benslimane, Jan Mendling, Nejib Ben Hadj-Alouane, Brian Blake, Olivier Perrin, Iman Saleh Moustafa, Sami Bhiri und Toumani Farouk. Bd. 8954. Lecture Notes in Computer Science. Cham: Springer, 2015, S. 96–107. ISBN: 978-3-319-22884-6. DOI: 10.1007/978-3-319-22885-3_9.

[58] Walter Oberschelp und Gottfried Vossen. *Rechneraufbau und Rechnerstrukturen (5. Aufl.)* 1992. ISBN: 978-3-486-22167-1. DOI: 10.1524/9783486595024.

[59] Nick Russell, Arthur Ter, David Edmond und Wil Aalst. „Workflow resource patterns“. In: (2005).

[60] W. M. P. van der Aalst, A. K. Alves de Medeiros und A. J. M. M. Weijters. „Process Equivalence: Comparing Two Process Models Based on Observed Behavior“. In: *Business process management*. Hrsg. von Schahram Dustdar. Bd. 4102. Lecture Notes in Computer Science. Berlin und Heidelberg: Springer, 2006, S. 129–144. ISBN: 978-3-540-38901-9. DOI: 10.1007/11841760_10.

[61] Michael Becker und Ralf Laue. „A comparative survey of business process similarity measures“. In: *Computers in Industry* 63.2 (2012), S. 148–167. ISSN: 01663615. DOI: 10.1016/j.compind.2011.11.003.

[62] Marcello La Rosa, Marlon Dumas, Chathura C. Ekanayake, Luciano García-Bañuelos, Jan Recker und Arthur H.M. ter Hofstede. „Detecting approximate clones in business process model repositories“. In: *Information Systems* 49 (2015), S. 102–125. ISSN: 03064379. DOI: 10.1016/j.is.2014.11.010.

[63] Ahmed Tealeb, Ahmed Awad und Galal Galal-Edeen. „Context-Based Variant Generation of Business Process Models“. In: *Enterprise, business-process and information systems modeling*. Hrsg. von Ilia Bider. Bd. 175. Lecture Notes in Business Information Processing. Heidelberg: Springer, 2014, S. 363–377. ISBN: 978-3-662-43744-5. DOI: 10.1007/978-3-662-43745-2_25.

[64] Lars Ackermann, Stefan Schönig und Stefan Jablonski. „Towards Simulation- and Mining-Based Translation of Process Models“. In: *Enterprise and Organizational Modeling and Simulation*. Hrsg. von Robert Pergl, Martin Molhanec, Eduard Babkin und Samuel Fosso Wamba. Bd. 272. Springer eBook Collection Computer Science. Cham: Springer, 2016, S. 3–21. ISBN: 978-3-319-49453-1. DOI: 10.1007/978-3-319-49454-8_1.

[65] Fabio Aiolli, Andrea Burattin und Alessandro Sperduti. „A Business Process Metric Based on the Alpha Algorithm Relations“. In: *Business Process Management Workshops*. Hrsg. von Florian Daniel, Kamel Barkaoui und Schahram Dustdar. Bd. 99. Lecture Notes in Business Information Processing. Berlin: Springer, 2012, S. 141–146. ISBN: 978-3-642-28107-5. DOI: 10.1007/978-3-642-28108-2_13.

[66] Andreas Schoknecht, Tom Thaler, Peter Fettke, Andreas Oberweis und Ralf Laue. „Similarity of Business Process Models—A State-of-the-Art Analysis“. In: *ACM Computing Surveys* 50.4 (2018), S. 1–33. ISSN: 0360-0300. DOI: 10.1145/3092694.

[67] Artem Polyvyanyy, Andreas Solti, Matthias Weidlich, Claudio Di Ciccio und Jan Mendling. „Monotone Precision and Recall Measures for Comparing Executions and Specifications of Dynamic Systems“. In: *ACM Transactions on Software Engineering and Methodology* 29.3 (2020), S. 1–41. ISSN: 1049-331X. DOI: 10.1145/3387909.

[68] Remco Dijkman, Marlon Dumas, Boudewijn van Dongen, Reina Käärik und Jan Mendling. „Similarity of business process models: Metrics and evaluation“. In: *Information Systems* 36.2 (2011), S. 498–516. ISSN: 03064379. DOI: 10.1016/j.is.2010.09.006.

[69] Boudewijn van Dongen, Remco Dijkman und Jan Mendling. „Measuring Similarity between Business Process Models“. In: *Active Flow and Combustion Control 2018*. Hrsg. von Rudibert King. Bd. 141. Notes on Numerical Fluid Mechanics and Multidisciplinary Design Ser. Cham: Springer International Publishing AG, 2019, S. 450–464. ISBN: 978-3-319-98176-5. DOI: 10.1007/978-3-540-69534-9_34.

[70] Jan Mendling, Boudewijn van F. Dongen und M. P. van der Aalst, Wil. „On the Degree of Behavioral Similarity between Business Process Models“. In: *6. Workshop der Gesellschaft für Informatik e.V. (GI)und Treffen ihres Arbeitskreises Gschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (WI-EPK)" St. Augustin, 29. November - 30. November 2007*. Hrsg. von Markus Nüttgens, Frank J. Rump, Andreas Gadatsch. CEUR Workshop Proceedings, 2007, S. 39–58.

[71] Carl Corea, Sabine Nagel, Jan Mendling und Patrick Delfmann. „Interactive and Minimal Repair of Declarative Process Models“. In: *Business Process Management Forum*. Hrsg. von Artem Polyvyanyy, Moe Thandar Wynn, Amy van Looy und Manfred Reichert. Bd. 427. Lecture Notes in Business Information Processing. Cham: Springer International Publishing

und Imprint Springer, 2021, S. 3–19. ISBN: 978-3-030-85439-3. DOI: 10.1007/978-3-030-85440-9_1.

[72] Paul Pichler, Barbara Weber, Stefan Zugal, Jakob Pinggera, Jan Mendling und Hajo A. Reijers. „Imperative versus Declarative Process Modeling Languages: An Empirical Investigation“. In: *Business Process Management Workshops*. Hrsg. von Florian Daniel, Kamel Barkaoui und Schahram Dustdar. Bd. 99. Lecture Notes in Business Information Processing. Berlin: Springer, 2012, S. 383–394. ISBN: 978-3-642-28107-5. DOI: 10.1007/978-3-642-28108-2_37.

[73] Jan Hidders, Marlon Dumas, Wil Aalst, Arthur Ter und Jan Verelst. „When Are Two Workflows the Same?“ In: *CATS, CRPIT* 41 (2005).

[74] Stefan Zugal, Jakob Pinggera und Barbara Weber. „Toward enhanced life-cycle support for declarative processes“. In: *Journal of Software: Evolution and Process* 24.3 (2012), S. 285–302. ISSN: 20477473. DOI: 10.1002/smr.554.

[75] Thomas H. Cormen, Charles Eric Leiserson, Ronald Linn Rivest und Clifford Stein. *Introduction to Algorithms*. third edition. The MIT Press Ser. Cambridge: MIT Press, 2009. ISBN: 9780262533058.

[76] Gaël Bernard und Periklis Andritsos. „Selecting Representative Sample Traces from Large Event Logs“. In: *ICPM, Eindhoven, Netherlands*. IEEE, 2021.

[77] Stefan Zugal, Jakob Pinggera und Barbara Weber. „The Impact of Testcases on the Maintainability of Declarative Process Models“. In: *Enterprise, business-process and information systems modeling*. Hrsg. von Terry Halpin. Bd. 81. Lecture Notes in Business Information Processing. Berlin und Heidelberg: Springer, 2011, S. 163–177. ISBN: 978-3-642-21758-6. DOI: 10.1007/978-3-642-21759-3_12.

[78] Gregory V. Bard. „Spelling-Error Tolerant, Order-Independent Pass-Phrases via the Damerau-Levenshtein String-Edit Distance Metric“. In: *IACR Cryptol. ePrint Arch.* (2006), S. 364.

[79] Leonid Boytsov. „Indexing methods for approximate dictionary searching“. In: *ACM Journal of Experimental Algorithmics* 16 (2011). ISSN: 1084-6654. DOI: 10.1145/1963190.1963191.

[80] Michiel Hazewinkel, Hrsg. *Encyclopaedia of mathematics*. Unabridged reprint of the original 10-vol. hardbound library ed. Dordrecht: Kluwer, 1995. ISBN: 1556080107.

[81] Vladimir I. Levenshtein. *Binary codes capable of correcting deletions, insertions, and reversals*. Bd. 4. Doklady Akademii Nauk SSSR. Band 163. 1966.

[82] Manuel Camargo, Marlon Dumas und Oscar González-Rojas. „Learning Accurate LSTM Models of Business Processes“. In: *Business Process Management*. Hrsg. von Thomas Hildebrandt, Boudewijn F. van Dongen, Maximilian Röglinger und Jan Mendling. Bd. 11675. Springer eBooks Computer Science. Cham: Springer, 2019, S. 286–302. ISBN: 978-3-030-26618-9. DOI: 10.1007/978-3-030-26619-6_19.

[83] Peter Hart, Nils Nilsson und Bertram Raphael. „A Formal Basis for the Heuristic Determination of Minimum Cost Paths“. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), S. 100–107. ISSN: 0536-1567. DOI: 10.1109/TSSC.1968.300136.

[84] G. Salton, A. Wong und C. S. Yang. „A vector space model for automatic indexing“. In: *Communications of the ACM* 18.11 (1975), S. 613–620. ISSN: 0001-0782. DOI: 10.1145/361219.361220.

[85] A. K. Alves de Medeiros, W.M.P. van der Aalst und A.J.M.M. Weijters. „Quantifying process equivalence based on observed behavior“. In: *Data & Knowledge Engineering* 64.1 (2008), S. 55–74. ISSN: 0169023X. DOI: 10.1016/j.datak.2007.06.010.

[86] Martin Käppel, Lars Ackermann, Stefan Schönig und Stefan Jablonski. „Language-independent look-ahead for checking multi-perspective declarative process models“. In: *Software & Systems Modeling* 20.5 (2021), S. 1379–1401. ISSN: 1619-1366. DOI: 10.1007/s10270-020-00857-8.

[87] Sven O. Krumke. *Graphentheoretische Konzepte und Algorithmen*. 3. Aufl. 2012. SpringerLink Bücher. Wiesbaden: Vieweg+Teubner Verlag, 2012. ISBN: 9783834822642. DOI: 10.1007/978-3-8348-2264-2.

[88] Manfred Reichert und Barbara Weber. *Enabling flexibility in process-aware information systems: Challenges, methods, technologies*. Berlin und Heidelberg: Springer, 2012. ISBN: 978-3-642-30408-8. DOI: 10.1007/978-3-642-30409-5.

[89] Sabine Nagel und Patrick Delfmann. „Investigating Inconsistency Understanding to Support Interactive Inconsistency Resolution in Declarative Process Models“. In: 2022.

[90] Dung My Thi Trinh, Amine Abbad-Andaloussi und Hugo A. López. „On the Semantic Transparency of Declarative Process Models: The Case of Constraints“. In: *COOPERATIVE INFORMATION SYSTEMS*. Hrsg. von Mohamed Sellami, Maria-Esther Vidal, Boudewijn van Dongen, Walid Gaaloul und Hervé Panetto. Bd. 14353. Lecture Notes in Computer Science. [S.l.]: SPRINGER INTERNATIONAL PU, 2023, S. 217–236. ISBN: 978-3-031-46845-2. DOI: 10.1007/978-3-031-46846-9_12.

[91] Amine Abbad Andaloussi, Jon Buch-Lorentsen, Hugo A. López, Tijs Slaats und Barbara Weber. „Exploring the Modeling of Declarative Processes Using a Hybrid Approach“. In: *Conceptual Modeling*. Hrsg. von Alberto H. F. Laender, Barbara Pernici, Ee-Peng Lim und José Palazzo M. de Oliveira. Bd. 11788. Springer eBooks Computer Science. Cham: Springer, 2019, S. 162–170. ISBN: 978-3-030-33222-8. DOI: 10.1007/978-3-030-33223-5_14.

[92] Amine Abbad Andaloussi, Andrea Burattin, Tijs Slaats, Anette Chelina Møller Petersen, Thomas T. Hildebrandt und Barbara Weber. „Exploring the Understandability of a Hybrid Process Design Artifact Based on DCR Graphs“. In: *Enterprise, Business-Process and Information Systems Modeling*. Hrsg. von Iris Reinhartz-Berger, Jelena Zdravkovic, Jens Gulden und Rainer Schmidt. Bd. 352. Springer eBooks Computer Science. Cham: Springer, 2019, S. 69–84. ISBN: 978-3-030-20617-8. DOI: 10.1007/978-3-030-20618-5_5.

[93] Johannes de Smedt, Jochen de Weerdt, Estefanía Serral und Jan Vanthienen. „Discovering hidden dependencies in constraint-based declarative process models for improving understandability“. In: *Information Systems* 74 (2018), S. 40–52. ISSN: 03064379. DOI: 10.1016/j.is.2018.01.001.

[94] Carl Corea, John Grant und Matthias Thimm. „Measuring Inconsistency in Declarative Process Specifications“. In: *Business Process Management*. Hrsg. von Claudio Di Ciccio, Remco Dijkman, Adela del Río Ortega und Stefanie Rinderle-Ma. Bd. 13420. Lecture Notes in Computer Science. Cham: Springer International Publishing und Imprint Springer, 2022, S. 289–306. ISBN: 978-3-031-16102-5. DOI: 10.1007/978-3-031-16103-2_20.

[95] Andreas Bauer, Martin Leucker und Christian Schallhart. „Runtime Verification for LTL and TLTL“. In: *ACM Transactions on Software Engineering and Methodology* 20.4 (2011), S. 1–64. ISSN: 1049-331X. DOI: 10.1145/2000799.2000800.

[96] Marco Montali, Maja Pesic, Wil M. P. van der Aalst, Federico Chesani, Paola Mello und Sergio Storari. „Declarative specification and verification of service choreographiess“. In: *ACM Transactions on the Web* 4.1 (2010), S. 1–62. ISSN: 1559-1131. DOI: 10.1145/1658373.1658376.

[97] J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, M. Woodger und Peter Naur. „Report on the Algorithmic Language ALGOL 60“. In: *Commun. ACM* 3.5 (1960), S. 299–314. DOI: 10.1145/367236.367262.

[98] Michael R. Garey und David S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. 27. print. A series of books in the mathematical sciences. New York u.a: Freeman, 1979. ISBN: 0716710455.

[99] Stephen A. Cook. „The complexity of theorem-proving procedures“. In: *Proceedings of the third annual ACM symposium on Theory of computing* (1971).

[100] Leslie Hall. „Computational Complexity“. In: *Encyclopedia of operations research and management science*. Hrsg. von Saul I. Gass und Michael Fu. Springer Reference. New York, NY: Springer Science + Business Media, 2013, S. 238–241. ISBN: 978-1-4419-1137-7. DOI: 10.1007/978-1-4419-1153-7_141.

[101] Claudio Di Ciccio, Mitchel H. M. Schouten, Massimiliano de Leoni und Jan Mendling. „Declarative Process Discovery with MINERful in ProM“. In: *BPM (Demos)*. 2015.

[102] Anti Alman, Claudio Di Ciccio, Dominik Haas, Fabrizio Maria Maggi und Alexander Nolte. „Rule mining with RuM“. In: *2nd International Conference on Process Mining, Italy*. 2020.

[103] Richard Dedekind. *Was Sind und Was Sollen Die Zahlen?* 9th ed. Wiesbaden: Springer Vieweg. in Springer Fachmedien Wiesbaden GmbH, 1961. ISBN: 9783663027881.

[104] Fabrizio Maria Maggi. „Declarative Process Mining with the Declare Component of ProM“. In: *Proceedings of the BPM Demo sessions 2013, Beijing, China, August 26-30, 2013*. Hrsg. von Marie-Christine Fauvet und Boudewijn F. van Dongen. Bd. 1021. CEUR Workshop Proceedings. CEUR-WS.org, 2013.

[105] Myriel Fichtner, Urs A. Fichtner und Stefan Jablonski. „An Experimental Study of Intuitive Representations of Process Task Annotations“. In: *Cooperative Information Systems*. Hrsg. von Mohamed Sellami, Paolo Ceravolo, Hajo A. Reijers, Walid Gaaloul und Hervé. Panetto. Bd. 13591. Lecture Notes in Computer Science. Cham: Springer International Publishing und Imprint Springer, 2022, S. 311–321. ISBN: 978-3-031-17833-7. DOI: 10.1007/978-3-031-17834-4_19.

[106] Kathrin Figl, Jan Mendling und Mark Strembeck. „Towards a Usability Assessment of Process Modeling Languages“. In: *CEUR Workshop Proceedings* 554 (2009).

[107] Gregor Jošt, Jernej Huber, Marjan Heričko und Gregor Polančič. „An empirical investigation of intuitive understandability of process diagrams“. In: *Computer Standards & Interfaces* 48 (2016), S. 90–111. ISSN: 09205489. DOI: 10.1016/j.csi.2016.04.006.

[108] Jan Mendling, Hajo Reijers und Jorge Cardoso. „What Makes Process Models Understandable?“ In: Bd. 4714. 2007. ISBN: 978-3-540-75182-3. DOI: 10.1007/978-3-540-75183-0_4.

[109] Constantin Houy, Peter Fettke und Peter Loos. „On the Theoretical Foundations of Research into the Understandability of Business Process Models“. In: *European Conference on Information Systems*. 2014.

[110] Dieter K. Tscheulin, Hrsg. *Gabler Lexikon Marktforschung*. Springer eBook Collection Business and Economics. Wiesbaden: Springer Gabler. in Springer Fachmedien Wiesbaden GmbH, 2004. ISBN: 9783322946355. DOI: 10.1007/978-3-322-94635-5.

[111] Fred Zijlstra und L. Doorn. „The Construction of a Scale to Measure Perceived Effort". In: *Department of Philosophy and Social Sciences* (1985).

[112] Eric M. Orendt, Myriel Fichtner und Dominik Henrich. „Robot programming by non-experts: Intuitiveness and robustness of One-Shot robot programming". In: *The 25th IEEE International Symposium on Robot and Human Interactive Communication*. Piscataway, NJ: IEEE, 2016, S. 192–199. ISBN: 978-1-5090-3929-6. DOI: 10.1109/ROMAN.2016.7745110.