Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Algorithmische Klassifikation der Layouts von BPMN-Diagrammen

Algorithmic Classification of Layouts of BPMN Diagrams

Masterarbeit

im Studiengang Informatik

von

Elias Aaron Baalmann

Prüfer: Prof. Dr. rer. nat. Kurt Schneider
Zweitprüfer: Dr. Daniel Lübke
Betreuer: Dr. Daniel Lübke

Hannover, 01.04.2022

Zusammenfassung

Da BPMN-Diagramme dazu genutzt werden, komplexe Zusammenhänge und Abläufe schnell und einfach nachzuvollziehen, ist deren Verständlichkeit essenziell. Das Layout eines Diagramms beeinflusst diese maßgeblich. Die aktuelle Forschung betrachtet hierbei unter anderem den Aspekt des Fluss-Layouts. Da unterschiedliche Fluss-Layouts zwar bezeichnet (z. B. Left-Right oder Multiline-Horizontal) aber nie formalisiert wurden, ist die Interpretation einzelner und der Vergleich zwischen Ergebnissen erschwert. In dieser Arbeit werden häufig genutzte Fluss-Layouts anhand eines großen GitHub Datensatzes identifiziert und formalisiert. Die hierfür festgelegte Hierarchie zerlegt das Fluss-Layout in Grundlayout, Orientierung und Variante. Im Anschluss wird basierend auf dieser Formalisierung ein Algorithmus konstruiert und eine Implementierung vorgestellt, um eine automatisierte Klassifikation von BPMN-Diagrammen zu ermöglichen. Hierfür werden alle schleifenfreien Pfade von einem Start- zu einem End-Objekt des Diagramms einzeln betrachtet und anhand der diskretisierten Sequenz-Fluss-Richtungen mit Hilfe von regulären Ausdrücken klassifiziert, bevor die einzelnen Pfad-Layouts zu einem Diagramm-Layout zusammengefasst werden. Die dargelegte Formalisierung stellt einen wichtigen Schritt in Richtung Standardisierung dar, die der aktuellen Praxis fehlt. Die präsentierte Automatisierung kann große Datensätze mit geringem Arbeitsaufwand klassifizieren, um z. B. Nutzerpräferenzen zu identifizieren. BPMN-Modellierern bietet sich weiterhin die Möglichkeit, eigene Diagramme auf Layout-Richtlinien hin zu überprüfen, die aufgrund der Formalisierung wiederum unmissverständlich definiert werden können.

Ein Ausschnitt dieser Arbeit wurde beim ZEUS 2022 Workshop eingereicht und angenommen [3]. Der dazugehörige Vortrag wurde mit dem „Best Presentation Award“ gewürdigt.

Abstract

BPMN diagrams are used to visualize complex processes in a way that allows readers to comprehend the underlying model quickly and easily. Thus, their understandability is a key requirement. Hence a diagram's understandability is strongly influenced by its layout, the aspect of flow layout gained considerable interest in the research community. But while different flow layouts have been identified (e.g., Left-Right or Multiline-Horizontal), currently there is no standardized formalization which significantly impedes the interpretation and comparability of results. In this thesis, prevalently used flow layouts are identified and formalized by utilizing a large GitHub dataset. Firstly, a hierarchy is introduced that splits the flow layout aspect into base structure, orientation, and variant. Subsequently, an algorithm based upon the aforementioned formalization is constructed and implemented so that BPMN diagrams are classified according to their flow layouts. This algorithm classifies each path of a diagram individually by matching sequences of the discretized sequence flow directions, after which the path layouts are aggregated to one diagram layout. The formalization is an important step towards standardization which is missing in current research practice. The provided automation enables easy analysis of large datasets which allows for many applications, e.g., to identify user preferences. Furthermore, practitioners can use the tool to validate their models against layout guidelines that in turn can be defined accurately using the formalized flow layouts.

An excerpt of this thesis was submitted to the ZEUS 2022 workshop. The paper was accepted [3] and the associated presentation honored with the "Best Presentation Award".

Danksagungen

An dieser Stelle möchte ich mich bei den Personen bedanken, die mir ermöglicht haben diese Masterarbeit fertig zu stellen, indem sie mich immer wieder motiviert, angeregt und unterstützt haben.

Besonderer Dank gebührt Daniel Lübke. Für die gute Betreuung und die Unterstützung bei dem zusammen beim ZEUS 2022 Workshop eingereichtem Paper möchte ich mich herzlich bedanken.

Ebenfalls möchte ich mich bei meinem Cousin Jonas Baalmann bedanken, der mich während der Arbeitszeit mehrfach durch Korrekturlesen unterstützt hat.

Inhaltsverzeichnis

1 Einleitung **1**

2 BPMN-Grundlagen **5**
2.1 Entwicklung der BPMN 5
2.2 BPMN-Diagrammtypen 6
2.3 BPMN-Elemente 7
2.4 Diagramm-Austauschbarkeit 8

3 Verwandte Arbeiten **11**
3.1 Layout von Graph-Darstellungen 11
3.2 Graph-Layout messen 11
3.3 BPMN-Layout 12
3.3.1 Layout-Richtlinien 12
3.3.2 Fluss-Layout 13
3.3.3 BPMN-Layout-Präferenz in großen Datensätzen 13

4 Identifikation und formale Definition von Fluss-Layouts **15**
4.1 Notwendigkeit der Formalisierung 16
4.2 Analysierter Datensatz 16
4.3 Vorgehensweise 16
4.3.1 Diskussion anhand eines Beispiels 17
4.3.2 Grundlegende Design-Entscheidungen 19
4.3.3 Konkretisierung von Fluss-Layouts mit Hilfe des Datensatzes 21
4.4 Layout-Hierarchie 22
4.4.1 Die sieben Grundlayouts 23
4.4.2 Untere Hierarchie-Ebenen: Orientierung und Variation 23
4.4.3 Das Straight Grund-Layout 24
4.4.4 Komplexe Grund-Layouts 26
4.5 Erzwungene Abweichungen vom Fluss-Layout 27
4.6 Formale Definition der Layout-Hierarchie 29
4.6.1 Vorbereitung 30
4.6.2 Definition des Pfad-Layouts 31

4.6.3 Definition des Diagramm-Layouts 41
4.6.4 Bestimmung angepasster Vektoren 41

5 Automatisierte Klassifikation des Fluss-Layouts 45
5.1 Algorithmus und Implementierung 45
5.1.1 BPMN-Modell lesen 46
5.1.2 Layoutpfade sammeln 46
5.1.3 Layoutpfad in Vektorkette umwandeln 47
5.1.4 Vektorkette vereinfachen 49
5.1.5 Vektorrichtung diskretisieren 50
5.1.6 Pfad-Layout bestimmen 51
5.1.7 Diagramm-Layout bestimmen 52
5.2 Nicht analysierbare Modelle 53
5.3 Nutzung der automatisierten Klassifikation 54

6 Validierung 55
6.1 Vergleich zu manueller Klassifikation 55
6.1.1 Ergebnisse der Klassifikationen 56
6.1.2 Gründe für Abweichungen der Klassifikationen 59
6.2 Anwendung auf großen Datensatz 64
6.3 Ergebnisse/ Diskussion 65

7 Fazit und Ausblick 69

A Reguläre Ausdrücke 71

B Verteilung der Fluss-Layouts 85

Kapitel 1

Einleitung

Business Process Model and Notation (BPMN) ist die Standardsprache für Geschäftsprozess-Modellierung [1, S. 10], die sowohl zu Dokumentations-, Kommunikations-, als auch zu Ausführungs-Zwecken verwendet wird. Sie ist ein Werkzeug, was ermöglichen soll, komplexe Zusammenhänge und Abläufe schnell und einfach nachzuvollziehen. Eine wichtige Anforderung an BPMN-Diagramme ist deshalb die Verständlichkeit des Modells. Konseqzenterweise ist BPMN-Verständlichkeit in den Fokus der Forschung gerückt. Zum Beispiel 2015 durch Corradini et al. [9], die Verständlichkeits-Richtlinien für BPMN-Diagramme in mehreren Kategorien vorstellen. Es werden unter anderem allgemeine Richtlinien, wie Modellgröße minimieren, und Richtlinien zum Aussehen der Diagramme, wie zum Beispiel das Vermeiden von Überlappungen der Elemente, genannt.

Ein Forschungszweig der BPMN-Verständlichkeits-Forschung betrachtet den Zusammenhang zwischen Diagramm-Layout und Diagramm-Verständlichkeit [32, 26, 39, 25]. Die zugrunde liegende Hypothese besagt, dass das Layout eines Diagramms einen großen Einfluss auf dessen Verständlichkeit hat. In der Vergangenheit wurden insbesondere relativ „lokale“ Metriken wie die Anzahl der Kantenkreuzungen (für BPMN Sequenz-Fluss-Kreuzungen) betrachtet. Neuerdings wurde jedoch immer mehr das Fluss-Layout untersucht. Der bisher ungenutzte Begriff „Fluss-Layout“ wurde in dieser Arbeit gewählt, um möglichst genau den Aspekt des Layouts zu beschreiben, der gemeint ist. Es geht sowohl um die Grundstruktur (z. B. gerade, mehrere Zeilen, U-förmig), als auch um die Orientierung dieser Struktur in einem Diagramm (bei gerade z. B. von links nach rechts, bei U-förmig z. B. Öffnung auf der rechten Seite). Dieser Aspekt wurde in ähnlicher Form schon unter den Namen Flow-Direction [15, 14] oder Layout-Direction [24, 25] betrachtet. Da mittlerweile auch komplexe Strukturen, wie zum Beispiel schlangenförmige (Snake) Layouts, untersucht werden, passen diese Termini nicht mehr. Die Entwicklung des Forschungsgebiets wird in Kapitel 3 genauer erläutert.

Problematisch ist im wissenschaftlichen Diskurs neben der uneindeutigen Definition des Themenbereichs, dass verschiedene Layouts nicht einheitlich voneinander abgegrenzt werden. Stattdessen werden Layouts nur durch Beispieldiagramme [25] oder vage Bezeichnungen wie „left-to-right“ [14] beschrieben. Dies erschwert a) die vorgestellten Erkenntnisse vollständig zu verstehen, b) die Ergebnisse zu replizieren und c) unterschiedliche Forschungsergebnisse zu vergleichen. So werden zum Beispiel Richtlinien angegeben, wie „Entsprechend dem Sequenzfluss [...] sind die Modelle von links nach rechts zu modellieren.“ [44, S. 9]. Welche Diagramme diese Richtlinie erfüllen ist unklar. Dies liegt daran, was genau „von links nach rechts zu modellieren“ bedeutet. Es könnte zum Beispiel heißen, dass jeder Sequenz-Fluss exakt nach rechts gerichtet sein sollte. Eine andere Interpretation wäre, dass die End-Objekte weiter rechts dargestellt werden sollten als die Start-Objekte, während die Position der restlichen Fluss-Objekte egal ist. Dies führt dazu, dass Modellierer nicht sicher stellen können, dass ihre Diagramme mit den Richtlinien des aktuellen Stands der Forschung übereinstimmen.

Ein Grund für die unklare Beschreibung der Layouts ist, dass es für Menschen in den meisten Fällen zwar einfach ist zu entscheiden, ob ein Diagramm ähnlich eines anderen gelayoutet ist, es aber schwer fällt, zu beschreiben, worin genau der Unterschied zwischen den Layouts besteht [4]. Dies gilt für alle Layout-Aspekte, insbesondere jedoch für das Fluss-Layout. Es zeigt sich, dass den meisten Diagrammen intuitiv ein Fluss-Layout zugewiesen und diese Klassifikation von Anderen ohne großes thematisches Verständnis nachvollzogen werden kann. Schwierigkeiten bestehen jedoch darin, diese Intuition genauer zu beschreiben, um eine objektive, regelbasierte Klassifikation zu ermöglichen. In dieser Arbeit wird genau dieses Problem betrachtet. Regelbasierte Klassifikation ist eminent, da die Klassifikation der Layouts andernfalls personenabhängig variieren kann. Zum Beispiel haben Lübke und Wutke 5297 Diagramme manuell klassifiziert und ca. 10% dieser Diagramme zunächst unterschiedlich eingestuft, bevor sie sich durch Diskussion auf eine Klassifikation einigen konnten [25]. Abbildung 1.1 zeigt ein sehr einfaches Diagramm, dessen Fluss-Layout

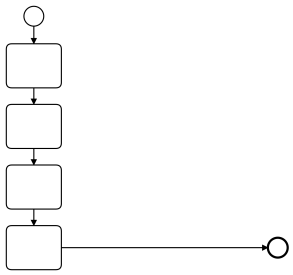

Abbildung 1.1: Diagramm mit unklarem Fluss-Layout

unterschiedlich interpretiert wird. So klassifizierte es einer der Autoren als Oben-Unten, während der andere es als nicht zuordenbar einstufte. Folglich ist es unklar, wie streng ein Diagramm einem bestimmten Fluss-Layouts entsprechen muss, um tatsächlich diesem Layout zugehörig zu sein.

Ein Ziel dieser Arbeit ist es, häufig verwendete Fluss-Layouts zu formalisieren. Dies ermöglicht es, eindeutig zu entscheiden, welches Layout für ein bestimmtes Diagramm verwendet wurde. Hierfür wird ein großer Datensatz von BPMN-Diagrammen analysiert, um möglichst viele Diagramme klassifizierbar zu machen. Die Betrachtung von Fluss-Layouts als Klassen, zu denen Diagramme zugeordnet werden können kann es nicht ermöglichen jedes mögliche Layout in beliebigem Detailgrad zu beschreiben, da unendlich viele Möglichkeiten existieren, ein BPMN-Diagramm darzustellen. Um dieser erzwungenen Einschränkung entgegenzuwirken, sollte die Formalisierung auf bisher nicht betrachtete Fluss-Layouts erweiterbar sein. Dies stellt sicher, dass die Formalisierung nicht exklusiv auf die, in dem einzelnen hier betrachteten Datensatz, vorhandenen Layouts beschränkt ist, sondern erweiterbar ist um allgemeingültig Fluss-Layouts zu beschreiben.

Allerdings sind durch eine Formalisierung der Fluss-Layouts nicht alle Probleme gelöst. Die manuelle Klassifikation von großen Datensätzen, wie von Lübke und Wutke durchgeführt [25], ist trotz Formalisierung fehleranfällig, da besonders die Beurteilung von Diagrammen mit vielen Elementen und komplexen Layouts aufwändig ist und menschliche Fehler unvermeidbar sind. Zusätzlich zur Fehleranfälligkeit ist die manuelle Klassifikation mit einem sehr großen Arbeitsaufwand verbunden. Es steht zu vermuten, dass dieser Aufwand durch die Formalisierung im Vergleich zur intuitiven Klassifikation noch erhöht werden würde. Da die Klassifikation von großen Datensätzen viele Einsatzmöglichkeiten hat, wird in dieser Arbeit aufbauend auf der Formalisierung ein Algorithmus zur automatisierten Klassifikation der Fluss-Layouts von BPMN-Diagrammen entwickelt. Zusätzlich wird ein Java-Werkzeug implementiert, dass diesen Algorithmus umsetzt. Dieses kann unter anderem von Forschern dafür genutzt werden Statistiken auf großen Datensätze zu erheben, um Fragen wie „Hängt das gewählte Fluss-Layout von der Leserichtung des Modellierers ab?“ zu beantworten. Außerdem kann es die Basis für ein Validierungs-Werkzeug liefern, das zum Beispiel als Plug-In von einem BPMN-Editor überprüft, ob eine Richtlinie zum Fluss-Layout eingehalten wurde.

In dieser Arbeit wird zunächst eine kurze Einführung zu BPMN gegeben (Kapitel 2) bevor die Entwicklung und der aktuelle Stand der Forschung des Themenbereichs in Kapitel 3 vorgestellt wird. In Kapitel 4 und 5 wird die Formalisierung und die darauf aufbauende automatisierte Klassifikation vorgestellt. Formalisierung und Automatisierung werden in Kapitel 6 validiert. Abschließend wird in Kapitel 7 zusammengefasst und ein Ausblick auf zukünftige Arbeiten gegeben.

Kapitel 2

BPMN-Grundlagen

Geschäftsprozesse sind eine Menge einer oder mehrerer Prozeduren oder Aktivitäten, die in einer vordefinierten Reihenfolge ausgeführt werden und zusammen normalerweise im Kontext einer Organisationsstruktur ein Ziel realisieren, wobei funktionale Rollen und Beziehungen definiert werden [41]. Geschäftsprozessmodellierung ist der Zeitraum, in dem Beschreibungen eines Geschäftsprozesses definiert oder bearbeitet werden. Diese Modellierung wird meist von Analysten und Managern genutzt um die Prozess-Effizienz und Qualität zu verbessern. BPMN ist aktuell der Standard für Geschäftsprozessmodellierung [7].

Um in den folgenden Kapiteln BPMN Diagramme analysieren zu können wird hier zunächst ein Überblick über die Modellierungsmöglichkeiten der BPMN gegeben. Hierfür wird die Geschichte der BPMN skizziert, gezeigt welche Diagrammtypen der BPMN in dieser Arbeit betrachtet werden und die wichtigsten Diagrammelemente vorgestellt, bevor auf die Serialisierung der Diagramme eingegangen wird, welche für die Deserialisierung, die für die automatisierte Klassifizierung benötigt wird, wichtig ist.

2.1 Entwicklung der BPMN

Die BPMN wurde von einem mehrheitlich aus Softwareunternehmen bestehenden Konsortium, der Business Process Management Initiative (BPMI), entwickelt. Die erste Spezifikation der BPMN wurde von einem Team unter der Leitung von Stephen A. White (IBM) entwickelt und 2004 veröffentlicht. Währenddessen ist die BPMI ein Teil der Object Management Group (OMG) geworden, eine Organisation, die für einige Softwarestandards wie der Unified Modeling Language (UML) bekannt ist. BPMN Version 1.0 wurde 2006 als OMG Standard akzeptiert. Die Versionen 1.1 und 1.2 brachten kleinere Änderungen mit sich, bevor 2011 die Version 2.0, mit umfassenden Änderungen und Erweiterungen, veröffentlicht wurde. Im Jahre 2013 wurde die BPMN ein offizieller ISO Standard (ISO 2013). [1, S. 10–11]

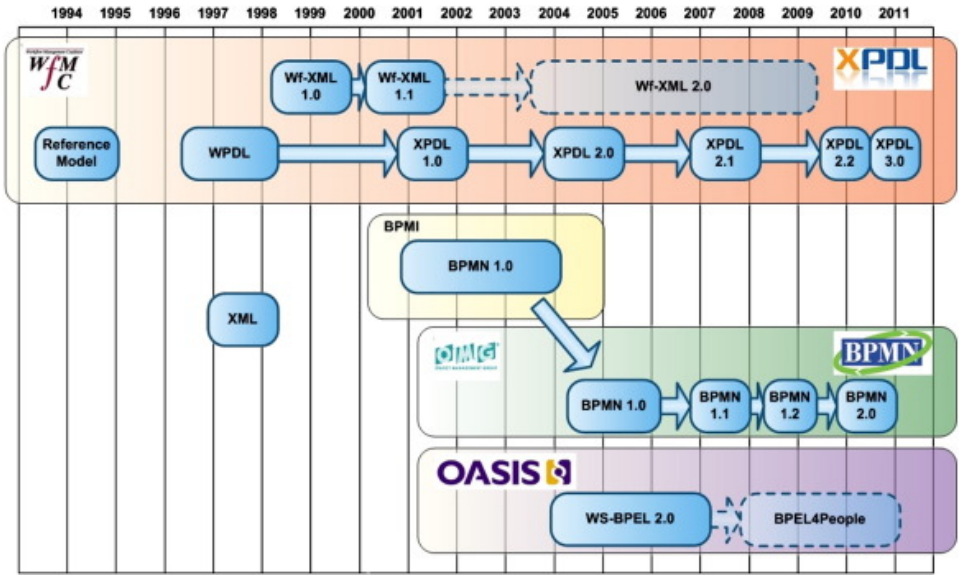

Abbildung 2.1: Entwicklungsgeschichte BPMN. Aus BPMN: An introduction to the standard. [7, S. 128]

Abbildung 2.1 zeigt diese Entwicklung zusammen mit den Entwicklungen der XML Process Definition Language (XPDL) und der WS-Business Process Execution Language (WS-BPEL).

In den 1.x Versionen stand die Abkürzung BPMN für Business Process Modeling Notation. In Version 2.0 wurde der Name zu Business Process Model **and** Notation geändert, um zu betonen, dass BPMN nicht nur die graphische Notation, sondern auch das Metamodel, das exchange format und die execution semantic umfasst. [1, S. 12]

2.2 BPMN-Diagrammtypen

Die 1.x BPMN-Versionen [28] können genutzt werden, um drei unterschiedliche Typen von Geschäftsprozessen zu modellieren. Hierfür werden drei Untermodelle verwendet. *Private Geschäftsprozesse* fokussieren interne organisationsspezifische Prozesse. Dies sind jene Prozesstypen, die generell mit Workflows oder BPM-Prozessen modelliert werden. *Abstrakte Prozesse* illustrieren Interaktionen zwischen einem privaten Geschäftsprozess und anderen Teilnehmern. Sie zeigen nur solche Aktivitäten, die an Interaktionen zwischen zwei oder mehr Teilnehmern beteiligt sind. *Kollaborationsprozesse* repräsentieren den Graph von Aktivitäten der unter anderem Nachrichtenaustauschpatterns zwischen zwei oder mehreren Businessprozessen beschreibt [7]. Obwohl die Untermodelle unterschiedliche Aspekte der Prozesse in den Vordergrund stellen, werden sie in Version 1.x mit dem gleichen

Diagrammtyp dargestellt.

Der aktuelle Standard ist BPMN 2.0. In dieser Version können drei Typen von Diagrammen unterschieden werden. *Prozess bzw. Kollaborationsdiagramme* werden wie BPMN 1.x Diagramme genutzt. Wenn private Prozesse dargestellt werden, wird das Diagramm oft Prozessdiagramm genannt, während Diagramme mit mehreren interagierenden Prozessen Kollaborationsdiagramme genannt werden. Die zwei neuen Diagrammtypen *Choreographiediagramme* und *Konversationsdiagramme* ermöglichen eine genauere Beschreibung der Interaktionen zwischen Teilnehmern, während private Prozesse in den Hintergrund rücken. In Choreographiediagrammen wird der Datenaustausch zwischen unterschiedlichen Teilnehmern in den Mittelpunkt gestellt, z. B., um komplexe Datenaustauschprotokolle darzustellen. Konversationsdiagramme bieten einen Überblick über die Teilnehmer und die Wechselbeziehungen zwischen ihnen, ohne den Datenaustausch im Detail zu beschreiben. [1, S. 11]

In dieser Arbeit werden Prozess bzw. Kollaborationsdiagramme betrachtet, da diese in der Praxis am häufigsten genutzt werden [1, S. 11].

2.3 BPMN-Elemente

Die fünf Kategorien von Elementen der BPMN sind Fluss-Objekte, Swimlanes, Verbindungs-Objekte, Daten und Artefakte. Fluss-Objekte können Ereignisse, Aktivitäten oder Gateways sein. Diese Objekte können über zwei Arten von Swimlanes gruppiert werden, durch Pools und Lanes. Außerdem werden Fluss-Objekte miteinander oder mit Daten und Artefakten wie Text-Annotationen über Sequenz-Flüsse, Nachrichten-Flüsse, Assoziationen oder Daten-Assoziationen verbunden. [29]

Eine Übersicht aller Elemente bietet das Poster der Berliner BPM-Offensive [5]. Für diese Arbeit wird jedoch nur ein Grundverständnis der wichtigsten Elemente und Syntaxregeln benötigt. Abbildung 2.2 zeigt ein Diagramm, das alle für das Verständnis wichtigen Elemente und Konzepte zeigt. Diagramme können mehrere Pools enthalten (Pool1 & Pool2), die wiederum in Lanes aufgeteilt werden können, wie bspw. Pool1 in Lane1 und Lane2. Fluss-Objekte können innerhalb von Pools durch Sequenz-Flüsse (z. B. a5 mit a4 oder a1 mit a5) und über Poolgrenzen hinweg durch Nachrichten-Flüsse (z. B. a5 mit a9) verbunden werden. Start-Objekte können entweder explizit durch Startereignisse (z. B. s1) oder implizit durch Aktivitäten ohne eingehende Sequenz-Flüsse (z. B. a8) dargestellt werden. Analog können auch End-Objekte entweder durch Endereignisse (z. B. e1) oder durch Aktivitäten ohne ausgehende Sequenz-Flüsse (z. B. a9) dargestellt werden. Aktivitäten können entweder Tasks (wie z. B. a3, a4 und a7) oder Sub-Prozesse (wie a2) sein. Verzweigungen können durch mehrere aus- bzw. eingehende Sequenz-Flüsse an einem Fluss-Objekt realisiert werden.

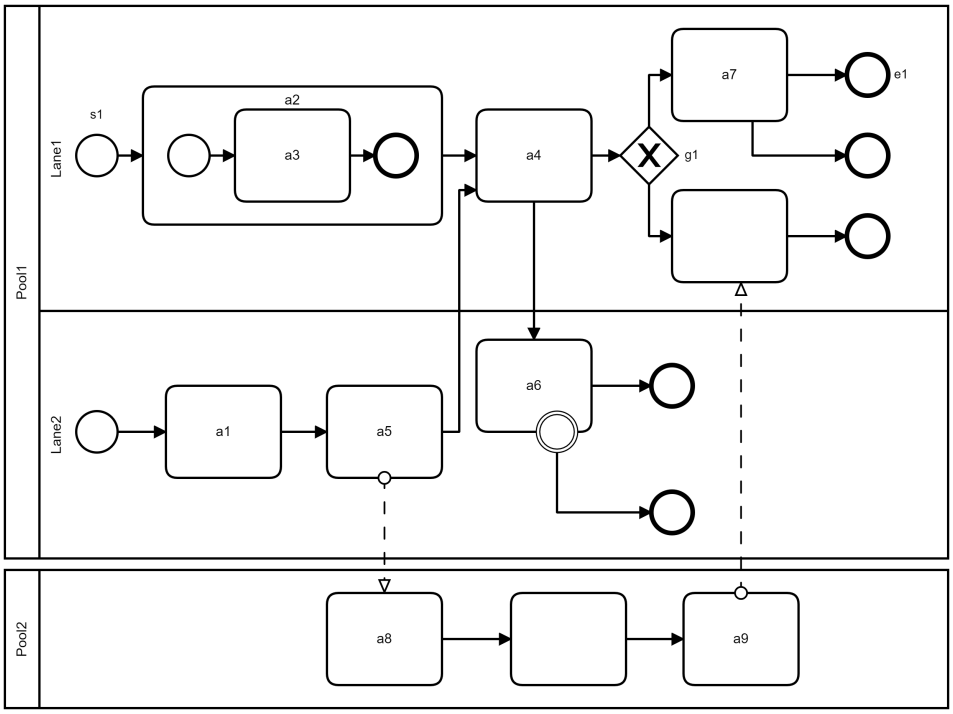

Abbildung 2.2: Beispieldiagramm zur Erklärung der Grundelemente

Hierfür können Gateways (wie g1) genutzt werden, wenngleich dies nicht zwingend erforderlich ist (siehe a4 & a7). Das Beispiel von a4 zeigt, dass Fluss-Objekte mehrere ein- und ausgehende Sequenz-Flüsse haben können. Ereignisse können an Aktivitäten angehängt werden (b1 ist an a6 angehängt). Vertiefend können die Regeln in der BPMN 2.0 Spezifikation nachgelesen werden. [29]

2.4 Diagramm-Austauschbarkeit

Die BPMN Tool Matrix [19] zählt aktuell 75 BPMN-Werkzeuge auf. Aufgrund dieser Menge von Editoren unterschiedlicher Anbieter, die zum Erstellen und Bearbeiten von BPMN-Diagrammen genutzt werden, ist die Austauschbarkeit von BPMN-Modellen zwischen den Werkzeugen zu einem wichtigen Thema geworden. In den BPMN-Versionen vor BPMN 2.0 wurde kein standardisierter Mechanismus zum Austausch von Diagrammen festgelegt. Stattdessen wird in der BPMN 1.2 Spezifikation [28] empfohlen BPMN-Modelle zu WS-BPEL 2.0 zu konvertieren, für das ein standardisiertes Austauschformat existiert. Da zwischen BPMN 1.2 und WS-BPEL 2.0 keine verlustfreie Konvertierung möglich ist, war diese Lösung keineswegs optimal [22].

BPMN 2.0 definiert ein eigenes XML-basiertes Austauschformat, welches Prozessmodelle in zwei Aspekte aufteilt: *Prozess-Modelle* beinhalten die Semantik und *Prozess-Diagramme* die visuelle Repräsentation des Prozess-Modells. Die Informationen zu Prozess-Diagrammen werden in der BPMN-Spezifikation auch BPMN Diagram Interchange (BPMN DI) genannt. Trotz der Standardisierung ist die Austauschbarkeit von BPMN-Diagrammen limitiert. Dies liegt zum einen daran, dass einige visuelle und semantische Aspekte nicht austauschbar sind, da z. B. nicht standardisiert ist wie die Farben von Elementen oder Text gespeichert wird. Zum anderen ist festzustellen, dass nicht alle Editoren dem Standard exakt folgen oder diesen unterschiedlich auslegen. Dies wird in Abschnitt 5.2 genauer betrachtet.

Kapitel 3

Verwandte Arbeiten

In diesem Kapitel werden die Entwicklung und der aktuelle Stand der Forschung im Bereich Graph-Layout, BPMN-Layout und Layout-Klassifizierung skizziert.

3.1 Layout von Graph-Darstellungen

Die Darstellung von allgemeinen Graphen sowie Graph basierten Diagrammen und deren Einfluss auf die Verständlichkeit der dargestellten Modelle wurde bereits in mehreren Arbeiten untersucht. Die Ausführungen von Helen Purchase können hierbei als treibende Kraft angesehen werden. So untersuchte sie mit unterschiedlichen Teams sogenannte *graph drawing aesthetics*, also darstellerische Aspekte, wie zum Beispiel die Anzahl von Kantenbiegungen oder Kantenkreuzungen, Orthogonalität oder Symmetrie. Für allgemeine Graphen wurde festgestellt, dass besonders Kantenkreuzungen vermieden werden sollten, um die Graphen möglichst leicht verständlich darzustellen [37, 32, 35].

Allerdings lassen sich nicht alle Ergebnisse, die für allgemeine Graph-Darstellungen gelten, auf Graph basierte Diagrammtypen wie BPMN- oder UML-Diagramme übertragen. Bspw. ist die Orthogonalität (Strukturieren der Knoten und Kanten anhand eines rechtwinkligen Rasters) der Diagramme für UML-Diagramme elementar, jedoch für allgemeine Graphen unwichtig [36]. Bei allgemeinen Graphen wird die Minimierung der Kantenbiegungen präferiert. Für UML-Diagramme ist dieser Aspekt unwichtiger als die Orthogonalität, sodass es sich lohnt Knicke in die Kanten einzubauen, um die Orthogonalität des Diagramms zu erhöhen [33, 34].

3.2 Graph-Layout messen

In dem 2002 veröffentlichten Artikel Metrics for Graph Drawing Aesthetics [31] stellt Purchase fest, dass zwar Richtlinien und sogar Graph-Layout-

Algorithmen existieren, die Aspekte jedoch unterschiedlich interpretiert werden. Um dem entgegenzuwirken werden Metriken vorgestellt, die messen, wie stark eine Graph-Darstellung einen Aspekt erfüllt. Hier wird unter anderem auch der Aspekt *consistend direction* (einheitliche Richtung) betrachtet. Die Metrik bestimmt hier, wie groß der Anteil der Kantensegmente ist, der exakt einer vorgegebenen Richtung entspricht. Einheitliche Richtung wird für UML-Diagramme, z. B. von Eichelberger und Schmid [13], als Richtlinie angegeben.

3.3 BPMN-Layout

Da BPMN-Diagramme Graph-basiert sind, lassen sich einige der Richtlinien für allgemeine Graphen, bzw. andere Graph-basierte Diagrammtypen wie UML, gut auf BPMN Diagramme übertragen. Allerdings ist zu beachten, dass BPMN-spezifische Anforderungen an Layout-Merkmale stellt. Unter anderem können die BPMN-Elemente unterschiedliche Größen haben. Außerdem müssen Partitionen wie Pools und Lanes beachtet werden und Beschriftungen in angemessener Schriftgröße Platz finden. [11, S. 33]

Folgend wird zunächst ein Überblick über BPMN-Layout-Richtlinien gegeben, bevor die Relevanz von Forschung zu Fluss-Layouts belegt und Anwendungsfälle für große Datensätze vorgestellt werden.

3.3.1 Layout-Richtlinien

BPMN-Diagramme stellen Prozesse dar, sodass die Diagramme einen Fluss beinahlten, der die Reihenfolge der Prozesselemente angibt. Anders als bei z. B. UML-Klassendiagrammen hat dieser Fluss-Aspekt einen großen Einfluss auf das Layout von BPMN-Diagrammen [12]. Das Fluss-Layout als Faktor für Diagramm-Verständlichkeit ist erst kürzlich in den Fokus der Forschung gerückt. So identifizierten Mendling et al. 2007 lediglich Faktoren wie Modellgröße, durchschnittlicher Verbindungsgrad (Anzahl der ein/ausgehenden Kanten je Knoten) oder die Dichte (Anzahl der Kanten im Verhältnis zur maximal möglichen Anzahl) des Modells [26]. 2009 benennen Schrepfer et al. mehrere Layout-Aspekte als relevanteste Faktoren für die Verständlichkeit [39]. Besonders wichtig seien Linienkreuzungen und Kantenbiegungen. Das Fluss-Layout wird auch hier nicht aufgeführt.

In der neusten Forschung zeichnet sich ab, dass das Fluss-Layout von BPMN-Diagrammen wichtig für dessen Verständlichkeit ist. Dies wird bei der Betrachtung von Layout-Richtlinien deutlich. Der Verein eCH gibt zum Beispiel in den BPMN-Modellierungskonventionen für die öffentliche Verwaltung an, dass BPMN-Diagramme entsprechend dem Sequenzfluss von links nach rechts zu modellieren sind [44]. Corradini et al. sind etwas vorsichtiger und empfehlen lediglich, die Einhaltung einer einheitlichen

Richtung, um lange dünne Modelle zu kreiren [9, 8]. Selbst die BPMN-Spezifikation empfiehlt, Modelle entweder von oben nach unten oder von links nach rechts darzustellen. Dies sei besonders wichtig, wenn sowohl Sequenz-Flüsse als auch Nachrichten-Flüsse genutzt werden. In dem Fall sollten die Nachrichten-Flüsse orthogonal zu den Sequenz-Flüssen ausgerichtet sein [29]. Die Präferenz von geraden Diagrammen in der Literatur zeigt sich auch in Beispieldiagrammen. So sind alle Diagramme in dem Praxishandbuch BPMN 2.0 von Freund und Rücker von links nach rechts orientiert [16]. Auch Werkzeuge, die automatisiert BPMN-Diagramme layouten, tun dies von links nach rechts [10, 21, 38].

Jeder azyklische Graph kann so dargestellt werden, dass alle Kanten von monoton nach oben gerichteten Linien repräsentiert werden [40]. Theoretisch könnte dementsprechend jedes BPMN-Modell, das keine Zyklen enthält durch ein Diagramm mit einer einheitlichen Layout-Richtung z. B. links nach rechts dargestellt werden. Allerdings haben Lübke et. al. in einer Eye Tracking Studie unter anderem gezeigt, dass Snake oder Multiline Layouts Scrolling vermeiden und daher für große Diagramme sinnvoll sind. [24]

3.3.2 Fluss-Layout

Obwohl das Fluss-Layout (oder ähnliche Aspekte wie Fluss-Richtung oder Layout-Richtung) als wichtiger Faktor für das Verständnis von Prozess-Modellen identifiziert wurde, fand laut Figl und Strembeck bis 2014 keine theoretische Diskussion des Themas statt [15]. 2015 stellten die gleichen Autoren fest, dass es vier Hauptoptionen für die Richtung eines Diagramms gibt: links nach rechts, oben nach unten, unten nach oben und rechts nach links. Allerdings erkennen sie auch an, dass komplexere Layouts aus den Grundrichtungen zusammengesetzt werden können [14]. Die Relevanz der Forschung zu Fluss-Layouts wird überdies durch Hypothesen bezüglich der Richtung von Diagrammen bestärkt. So wird angenommen, dass die präferierte Richtung von Diagrammen von der Muttersprache des Nutzers abhängt [27]. Bspw. sollen Japaner aufgrund der höheren Vertikalen Komponente im der japanischen Schriftbild besser mit nach unten gerichteten Diagrammen arbeiten können als Australier, die wiederum nach rechts gerichtete Layouts präferieren [17]. Zu ähnlichen Ergebnissen kommen Tversky et al., die feststellen, dass Englisch sprechende Kinder eher von links nach rechts „denken“ als Kinder, die Hebräisch sprechen [43].

3.3.3 BPMN-Layout-Präferenz in großen Datensätzen

Um die Layout-Gewohnheiten von Modellierern zu untersuchen, sind große Datensätze hilfreich, die Diagramme von vielen unterschiedlichen Nutzern enthalten.

Analysen von großen Datensätzen wurde zum Beispiel durch Projekte

wie BPMN in the Wild ermöglicht [18]. Hierbei wurden 6.163.217 Repositories (10% der damals auf GiHub.com vorhandenen Software-Repositories) untersucht. Es wurden 21306 potenzielle BPMN-Prozessmodelle in 1251 Repositories gefunden. Durch Filterung nach BPMN 2.0's XML-basierter Serialisierung blieben 16907 und nach Entfernen der Duplikate 8904 Modelle übrig. Neuerdings wurde dieser Ansatz erweitert, um alle 82,8 Millionen Projekte auf GitHub abzudecken [42]. Dadurch konnte der Datensatz auf 79.713 verschiedene Diagramme vergrößert werden. Die Verallgemeinerung von Erkenntnissen, die auf GitHub-Datensätzen gewonnen wurden, ist jedoch nicht immer sinnvoll, unter anderem, da die meisten GitHub Projekte privat und inaktiv sind [20]. Allerdings bieten sie besonders aufgrund der Stichprobengröße viele Möglichkeiten. Diese Arbeit basiert auf ebensolchen GitHub Datensätzen, um häufig genutzte Fluss-Layouts zu identifizieren (siehe Abschnitt 4.3.3) und um die entwickelte Automatisierung, hinsichtlich ihrer Fähigkeit große Datensätze zu analysieren, zu validieren (siehe Abschnitt 6.2).

Wie erwähnt scheinen gerade, insbesondere von links nach rechts gerichtete Layouts, am häufigsten empfohlen und genutzt zu werden. Wie oft bestimmte Layouts tatsächlich genutzt werden ist unklar. Effinger et al. behaupten zum Beispiel, die Fluss-Richtung sei in BPMN-Diagrammen meistens oben nach unten oder links nach rechts, belegen dies aber nicht [12]. Erst 2021 veröffentlichten Lübke und Wutke den Beitrag Analysis of Prevalent BPMN Layout Choices on GitHub [25], in dem sie die, in dem BPMN in the Wild Datensatz genutzten, Fluss-Layouts (da Layout-Richtungen genannt) untersuchen. Hierfür klassifizierten die Autoren manuell 5297 Diagramme, indem sie jedem Diagramm entweder eines der sechs betrachteten Fluss-Layouts (Left-Right, Top-Down, Snake-Horizontal, Snake-Vertical, Multiline-Horizontal und Multiline-Vertical) zuordneten, oder es als nicht klassifizierbar einstuften. Unter anderem wurde die Verteilung der Fluss-Layouts betrachtet. Es wurden ca. 82% als Left-Right, 10% als nicht klassifizierbar und 6% als Top-Down eingeordnet.

Kapitel 4

Identifikation und formale Definition von Fluss-Layouts

Für eine sinnvolle Klassifikation gilt es zunächst, ein einheitliches Verständnis für die Unterschiede distinkter Fluss-Layouts zu schaffen. Bernstein und Soffer stellten 2015 fest, dass, obwohl Unterschiede zwischen BPMN-Layouts intuitiv klar sind, es schwierig ist zu beschreiben, worin genau die Distinktion besteht. Problematisch ist, dass präzise Konzepte fehlen, um Diagramm-Layouts und eingehergehend Unterschiede zwischen diesen zu beschreiben [4]. Die Autoren sammelten in einer Studie Layout-Merkmale und fassten diese in sieben Gruppen zusammen. Eine dieser Gruppen ist die „Richtung“, für die bspw. das Merkmal der generellen Richtung des Diagramms genannt wird. Die Autoren unterscheiden hier zwischen horizontal (links nach rechts oder rechts nach links) und vertikal (oben nach unten oder unten nach oben). Die Autoren stellen außerdem fest, dass es Richtungsänderungen innerhalb eines Diagramms geben kann, z. B., um größere Modelle auf einer Seite darstellen zu können. Dieser Umstand wurde in der Studie durch Aussagen wie „Dieses Modell sieht aus wie eine Treppe“ betont. [4]

In dieser Arbeit wird eine detailliertere Unterscheidung der Layouts angegeben. Indem nicht nur zwischen horizontalen und vertikalen Richtungen unterschieden wird, sondern auch komplexe Fluss-Layouts definiert werden ist eine genaue Beschreibung des Layouts von vielen BPMN-Diagrammen möglich.

Um die getroffene Unterscheidung nachvollziehbar darzustellen wird hier zunächst genauer erklärt, warum die formale Definition der Fluss-Layouts nötig ist (Abschnitt 4.1) und ein großer Datensatz eingeführt (Abschnitt 4.2), mit dessen Hilfe die zugrunde liegende Methodologie erläutert wird (Abschnitt 4.3). Nachfolgend werden Entscheidungen bezüglich betrachteter Layouts (Abschnitt 4.4), aufgetretener Sonderfälle (Abschnitt 4.5) und der konkreten Formalisierung der einzelnen Fluss-Layouts (Abschnitt 4.6) konkretisiert.

4.1 Notwendigkeit der Formalisierung

Zusätzlich zu der Schwierigkeit, Layouts zu beschreiben, wird die Kommunikation über dieses Thema durch fehlende Standardisierung und Formalisierung erschwert. So nutzen bspw. Figl und Strembeck den Begriff „left-to-right“ für nach rechts gerichtete Teile eines Diagramms, und sprechen von Richtungswechseln innerhalb von Diagrammen. so werde zum Beispiel die right-to-left Richtung meist im Kontext von Schleifen genutzt [14]. Dem gegenübergestellt betrachten Lübke und Wutke die „Layout-Richtung“ als globales Attribut eines Diagramms, indem sie jedem Diagramm genau eine Richtung zuweisen [25]. Entsprechend betrachteten die Autoren unter anderem eine „Snake-Horizontal“-Layout-Richtung. Für Figl und Strembeck wäre solch ein Layout wohl eine Zusammensetzung aus mehreren Richtungen. Den beiden Arbeiten ist gemein, dass nicht klar beschrieben wird, welche Anforderungen ein Diagramm erfüllen muss, um einer bestimmten Richtung zugeordnet werden zu können. Um diesen Unklarheiten entgegenzuwirken und einer Standardisierung näher zu kommen, werden in dieser Arbeit spezifische Charakteristika distinkter Fluss-Layouts klar beschrieben.

4.2 Analysierter Datensatz

Diese Arbeit basiert auf einer großen Stichprobe, mit deren Hilfe z. B., oft genutzte Fluss-Layouts identifiziert werden (siehe Abschnitt 4.3.3). Dieser Datensatz beinhaltet 67169 Dateien, die über die GitHub code search API v3 gefunden wurden. Für 53984 dieser Modelle konnten mittels des bpmn-to-image Tools von BPMN-iO [6] PNG-Dateien generiert werden. Diese können genutzt werden, um Diagramme manuell zu analysieren. Aus verschiedenen Gründen sind einige der Modelle nicht klassifizierbar (siehe Abschnitt 5.2). 48675 Modelle lassen sich automatisiert klassifizieren.

4.3 Vorgehensweise

Um die objektive Klassifikation der Fluss-Layouts gewährleisten zu können, müssen für mehrere Aspekte Regeln definiert werden. Unter anderem sollten die folgenden Fragen beantwortet werden.
1. Welche Eigenschaften haben bestimmte Fluss-Layouts?
2. Welcher „Spielraum“ sollte innerhalb von Fluss-Layouts möglich sein?
3. Welches Diagramm-Layout liegt vor, wenn die einzelnen Pfade durch das Diagramm unterschiedliche Layouts haben?
4. Können Diagramme, die aus mehreren Fluss-Layouts zusammengesetzt sind, klassifiziert werden?

Diese Fragen werden subjektiv unterschiedlich beantwortet, sodass eine objektive Klassifikation unmöglich wird. Dies zu ändern kann auf mehrere

Arten und Weisen erreicht werden. Eine Möglichkeit wäre eine Umfrage, in denen die Teilnehmer eine Vielzahl von Diagrammen zuordnen. Die resultierende Formalisierung bestünde darin, dass möglichst viele Teilnehmer mit der objektiven Klassifikation übereinstimmen. Für diese Arbeit wurde sich allerdings bewusst gegen diese Vorgehensweise entschieden, da die Bestimmung, welches Fluss-Layout vorliegt, für einige Diagramme so schwierig ist, dass ein tieferes Verständnis der Thematik vorliegen muss als von Teilnehmer*innen einer solchen Umfrage vorausgesetzt werden kann. Deshalb wurden die schwierigen Fälle mit dem Ziel diskutiert, eine Formalisierung der Fluss-Layouts zu finden, die in sich einheitlich ist und bei der jede Entscheidung begründet werden kann. Dies bedeutet nicht, dass die getroffen Entscheidungen exklusiv richtig sind, sondern nur, dass sie einer reproduzierbaren Logik folgen und deshalb nachvollziehbar sind.

Im Folgenden wird anhand eines Beispiels der Entscheidungsprozess hinter der Kategorisierung von Fluss-Layouts demonstriert, um der Formalisierung der Fluss-Layouts näher zu kommen. Im Anschluss wird die konkrete Vorgehensweise zur Bildung und Abgrenzung der Fluss-Layouts beschrieben.

4.3.1 Diskussion anhand eines Beispiels

Abbildung 4.1 zeigt sieben Diagramme, anhand derer die Abgrenzung zwischen Fluss-Layouts erläutert werden kann. So könnte argumentiert werden, dass alle Diagramme das Fluss-Layout Snake-Vertical haben, ab hier wegen der größeren Präzision Snake-SE (SE steht für South-East und beschreibt die Orientierung des Snake Fluss-Layouts) genannt, aufweisen. Allerdings wird klar, dass zwar Diagramm A eindeutig Snake-SE ist, Diagramm G aber kaum diesem Layout zugeordnet werden sollte.

Diagramm B ist identisch zu A, es hat lediglich eine Zeile weniger. Diagramm C hat im Vergleich zu B einen größeren Zeilenabstand. Diagramm D hat eine zusätzliche Aktivität zwischen den zwei Zeilen, E hat keine Aktivitäten auf den Zeilen, F verkürzt die Zeilen und G ist noch weiter horizontal gestreckt mit mehr Aktivitäten auf dem horizontalen Pfad.

Es stellt sich die Frage, welches Diagramme nicht Snake-SE sind und welche Layouts stattdessen vorliegen. Um dem Diagramm G ein Fluss-Layout zuzuordnen, müssen mehrere Aspekte abgewogen werden. Eine Möglichkeit ist festzulegen, dass das Diagramm das Layout Left-Right, ab hier Straight-E genannt, aufweist, und lediglich etwas unsauber gezeichnet wurde. Andererseits könnten auch Argumente dafür gefunden werden, dass das Fluss-Layout ein „U“ darstellt, oder aber, dass dieses Diagramm wegen seinem speziellem Layout keinem Idealtypus zugeordnet werden kann. Für diese Arbeit wurde die Entscheidung wie folgt getroffen: Da Diagramme wie das Diagramm G, bei denen nur das Start und/oder End-Objekt nicht auf der geraden Linie der anderen Elemente liegt, relativ häufig vorkommen, sollten sie einem Fluss-Layout zugehörig sein. Allerdings ist der Unterschied

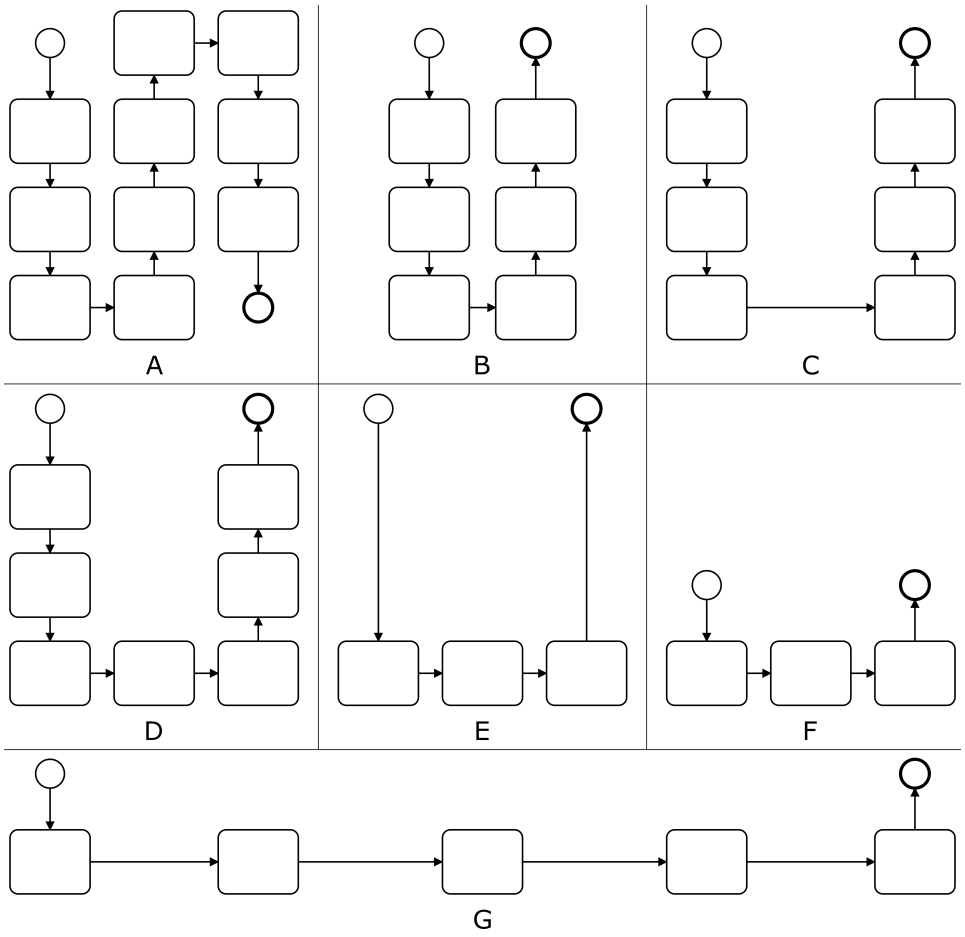

Abbildung 4.1: Beispieldiagramme für Snake-SE-Clean (A, B, C), U-SE-Clean (D), Straight-E-U (E, F, G)

zu einem sauberen Straight-E Layout zu klein, um eine komplett neue Klasse von Layouts zu definieren und gleichzeitig zu groß, um sie mit sauberen Straight-E Diagrammen einer Klasse zuzuordnen. Deshalb wurde eine Hierarchie von Fluss-Layouts festgelegt. Straight-E ist demnach nicht mehr nur ein Fluss-Layout, sondern eine Kategorie von mehreren leicht unterschiedlichen Layouts. Diagramm E, F und G haben gemeinsam, dass sie eine gerade Kette von mehreren Elementen von links nach rechts bilden und lediglich die Start- bzw. End-Objekte nicht auf dieser Linie liegen. Diese Diagramme haben das Fluss-Layout Straight-E-U. Zwar hat Diagramm D die gleiche Form wie Diagramm E, jedoch unterscheiden sie sich fundamental, da nicht nur die Start- und End-Objekte nicht auf der geraden Linie liegen. Diagramm D ist stattdessen in drei Ketten aufgeteilt: eine ist gerade nach unten, eine ist nach rechts und eine ist nach oben gerichtet. Aus diesem Grund kann zwar Diagramm E, aber nicht Diagramm G, dem Fluss-Layout

Straight-E-U, zugeordnet werden.

Erneut stellt sich die Frage nach dem Umgang mit ambivalenten Zuordnungsmöglichkeiten. Aus den sechs von Lübke und Wutke betrachteten Fluss-Layouts ist wohl Snake-SE (von den Autoren Snake-Vertical genannt) die beste Wahl für Diagramm D, allerdings würde diese Zuordnung einige Fragen aufwerfen. Zum Beispiel, ob es egal ist, wie viele Elemente auf den Verbindungen zwischen den vertikalen Linien liegen. Diagramm G veranschaulicht, dass eine entsprechende Zuordnung nicht logisch wäre. Selbst wenn Diagramm G noch um Elemente auf den vertikalen Linien erweitert würde, und somit nicht mehr Straight-E-U zugeordnet werden könnte, wäre es kaum ein Snake-SE Layout. Da Diagramme mit ähnlichen Layouts mehrfach vorkommen, wird die neue Fluss-Layout-Kategorie **U** eingeführt. Diagramm C wird trotz eines großen Abstandes zwischen den vertikalen Linien, ebenso wie A und B, dem Snake-SE Layout zugeordnet. Letztendlich folgt die Zuordnung
A, B, C → Snake-SE-Clean
D → U-SE-Clean
E, F, G → Straight-E-U.

In der oben durchgeführten Diskussion sind einige Prinzipien, nach der die Konstruktion der Fluss-Layouts in dieser Arbeit vollzogen wird, klar geworden. So wird für die Unterscheidung von Fluss-Layouts weniger die tatsächliche Form des Diagramms, sondern dessen Zusammensetzung aus Folgen von Elementen, die in gleiche Richtungen verlaufen, berücksichtigt.

4.3.2 Grundlegende Design-Entscheidungen

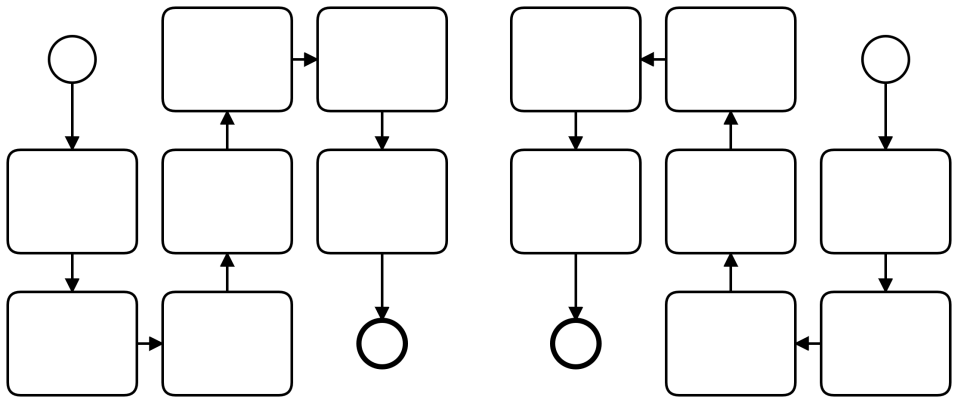

Abbildung 4.2: Snake-SE (links), Sake-SW (rechts)

Bevor festgelegt werden kann, welche Fluss-Layouts betrachtet werden, müssen einige Grundlegende Entscheidungen getroffen werden. Wie erwähnt gibt es im wissenschaftlichen Diskurs unterschiedliche Betrachtungsweisen

des Themenkomplexes. In dieser Arbeit wird das Fluss-Layout Lübke und Wutke folgend und damit entgegen Figl und Strembeck, als globales Attribut eines Diagramms verstanden. Auf diese Weise können Diagramme leichter nach ihrem Fluss-Layout gruppiert und die Layout bestimmenden Spezifika klarer definiert werden. So würde bei einer lokalen Betrachtung des Fluss-Layouts ein Diagramm mit dem oben gezeigten Snake-SE Layout zum Beispiel mit *„nach-unten, nach-oben, nach-unten,...“* beschrieben werden. Dies ließe keine Unterscheidung zwischen Snake-SE und Snake-SW zu, da die beiden in Abb. 4.2 gleich bezeichnet werden würden. Um diese Mehrdeutigkeit zu vermeiden könnten die Kanten zwischen den Spalten mit in die Beschreibung aufgenommen werden. Snake-SE würde dementsprechend bspw. mit *„nach-unten, nach-rechts, nach-oben,...“* beschrieben werden. Allerdings wären auf diese Weise zum Beispiel Diagramm B und Diagramm D aus Abb. 4.1 nicht auseinander zu halten, da beide mit „nach-unten, nach-rechts, nach-oben“ beschrieben werden würden.

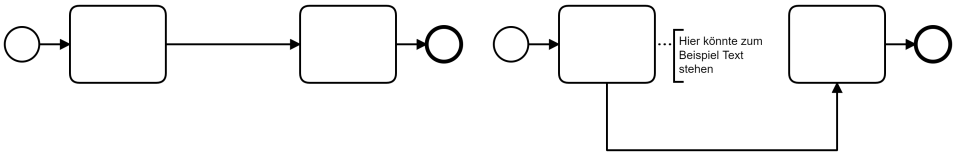

Abbildung 4.3: Zwei bezüglich ihres Fluss-Layouts (Straight-E-Clean) äquivalente Diagramme. Links ohne, rechts mit Knicken in Sequenz-Fluss.

BPMN-Modelle unterscheiden zwischen zwei Fluss-Arten. Zum einen gibt es den Sequenz-Flüsse, die die Ausführungsreihenfolge der Elemente festlegen und zum anderen Nachrichten-Flüsse, die die Übermittlung von Daten visualisieren. Wie in den meisten früheren Arbeiten [44, 8, 25] wird auch in dieser Arbeit nur der Sequenz-Fluss für die Bestimmung des Fluss-Layouts verwendet. Allerdings heißt das nicht, dass das Layout der einzelnen Sequenz-Flüsse für das Fluss-Layout relevant sind. So sind die Diagramme in Abb. 4.3 bezüglich ihres Fluss-Layouts identisch, obwohl der Pfeil, der den mittleren Sequenz-Fluss darstellt, im zweiten Diagramm nicht gerade nach rechts gerichtet ist. Für das Fluss-Layout ist nur relevant, in welcher Struktur die Elemente, auf dem Pfad von Start- zu End-Objekt der durch die Sequenz-Flüsse gebildet wird, angeordnet sind.

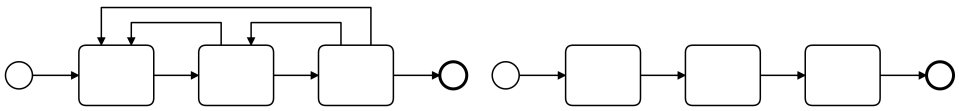

Abbildung 4.4: Zwei bezüglich ihres Fluss-Layouts äquivalente Diagramme, einmal mit, einmal ohne Rückwärtskanten

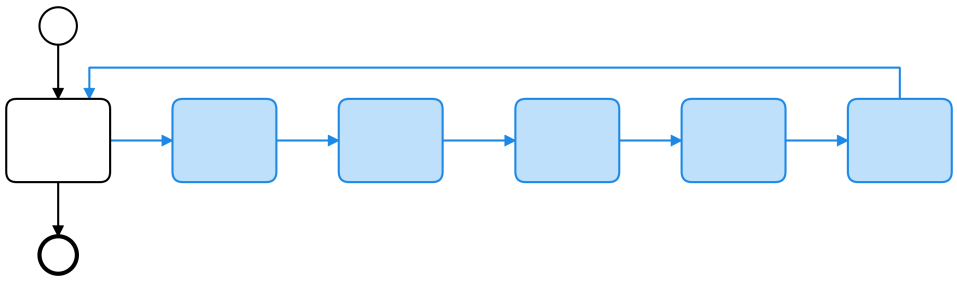

Abbildung 4.5: Diagramm mit Großteil der Fluss-Elemente in einer Schleife

Des Weiteren wird für diese Arbeit festgelegt, dass nur die Wege von einem Start- zu einem End-Objekt für das Fluss-Layout relevant sind, die keine Schleifen enthalten. In Abb. 4.4 wird dies anhand zweier Diagramme verdeutlicht, die beide als Straight-E klassifiziert werden. Dies schränkt die Aussagekraft des Fluss-Layouts für Diagramme, bei denen ein Großteil der Fluss-Elemente nur über Schleifen erreichbar ist, ein. Das Diagramm aus Abb. 4.5 veranschaulicht dieses Problem. Die Markierten Fluss-Elemente und Sequenzflüsse werden für die Bestimmung des Fluss-Layouts nicht betrachtet. Das Diagramm hat also nach Definition das Fluss-Layout Straight-S (gerade nach unten). Es gibt Ansätze, diese Fälle anders zu behandeln, zum Beispiel, indem für Wege, die durch Schleifen laufen, die Sequenz-Flüsse umgekehrt und die resultierenden Layouts am Ende um 180° gedreht werden. Da dies jedoch mit einigen Schwierigkeiten verbunden ist und nur relativ wenige Diagramme anders klassifiziert würden, wurde für diese Arbeit darauf verzichtet.

4.3.3 Konkretisierung von Fluss-Layouts mit Hilfe des Datensatzes

Die Vorgehensweise, um zu entscheiden, welche Fluss-Layouts zu betrachten sind, basiert auf dem zuvor beispielhaft gezeigten Prinzip der Diskussion. Der Ablauf ist in Abbildung 4.6 dargestellt. Es wird das in dieser Arbeit vorgestellte Werkzeug zur automatisierten Klassifikation genutzt. Dieses wird iterativ erweitert, bis alle Diagramme entweder klassifiziert werden können oder entschieden wird, dass das Diagramm ein zu spezielles Fluss-Layout verwendet, um klassifiziert zu werden. Hierfür werden zunächst alle Diagramme des großen GitHub Datensatzes automatisiert klassifiziert. Danach wird eines der nicht Klassifizierten Diagramme ähnlich wie oben diskutiert, um zu entscheiden, ob das Diagramm:

1. keinem der bis jetzt betrachteten Fluss-Layouts zuordenbar ist, es aber ein Fluss-Layout nutzt, welches klassifizierbar sein sollte

2. einem vorhandenem Fluss-Layout zugeordnet werden kann

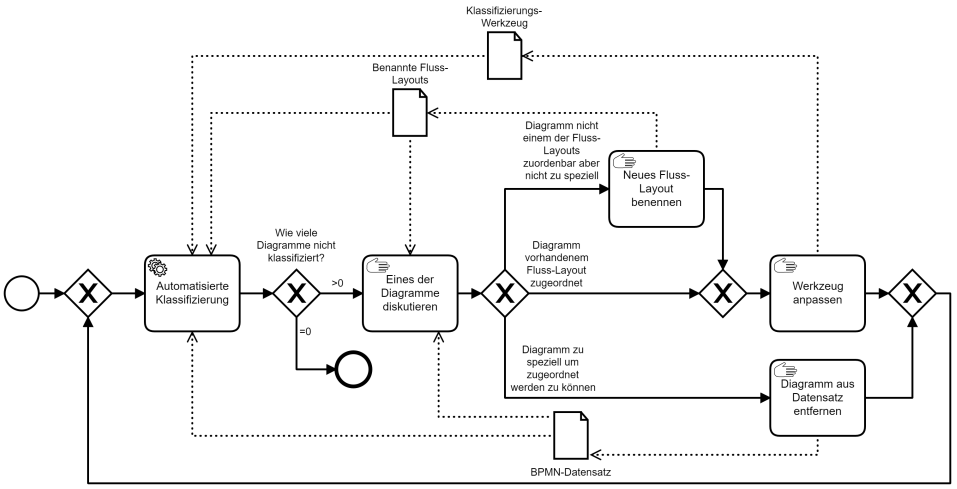

Abbildung 4.6: Vorgehensweise zum Identifizieren interessanter Fluss-Layouts

3. ein Fluss-Layout verwendet, das zu speziell ist um es hier zu betrachten

Im 1. Fall wird das neue Layout benannt und zu den betrachteten Fluss-Layouts hinzugefügt, bevor das Werkzeug zur automatisierten Klassifikation angepasst wird, sodass es dieses Fluss-Layout zukünftig erkennt. Im 2. Fall wird ebenfalls das Werkzeug angepasst, sodass das betrachtete Diagramm und Diagramme mit sehr ähnlichen Layouts in Zukunft dem richtigen Fluss-Layout zugeordnet werden können. Da im dritten Fall festgestellt wird, dass das Diagramm-Layout nicht betrachtet werden soll, wird es aus dem Datensatz entfernt. Diese Schritte werden so lange wiederholt, bis alle (noch im Datensatz vorhandenen) Diagramme klassifiziert werden können. Da möglicherweise der Eindruck entsteht, der Vorgang müsse für jedes einzelne der ca. 54000 Diagramme ausgeführt werden, hier nochmal die Klarstellung: Der Unterschied zu einer vollständigen manuellen Klassifikation des Datensatzes besteht darin, dass eine Werkzeug Anpassung die Klassifikation mehrerer neuer Diagramme erlaubt.

4.4 Layout-Hierarchie

Die grundlegende Idee von Fluss-Layouts basiert der Methode, ähnliche Diagramm-Layouts zu gruppieren und zu benennen. Diese Gruppierung kann mit unterschiedlichem Detaillierungsgrad geschehen. Während eine detaillierte Variante sehr viele unterschiedliche Fluss-Layouts betrachten würde, kann eine nicht so detaillierte Gruppierungen nur wenige Layouts unterscheiden. Beide Optionen haben Vor- und Nachteile. Ein Vorteil von wenigen Fluss-Layouts ist, dass schnell ein intuitives Verständnis durch

wenige Beispiel-Diagramme vermittelt werden kann. Viele Fluss-Layouts hingegen bieten die Möglichkeit, das Layout von vielen unterschiedlichen Diagrammen genau zu beschreiben. Um die Vorteile von beiden Varianten zu kombinieren, wurde für diese Arbeit eine Fluss-Layout-Hierarchie erstellt. Auf diese Weise können Diagramme nicht nur sehr detailliert klassifiziert werden, sondern mit Hilfe der oberen Kategorien auch einen schnellen Überblick bieten. Die erstellte Hierarchie wird im Folgenden vorgestellt.

4.4.1 Die sieben Grundlayouts

Auf der obersten Ebene der Hierarchie stehen sieben Grund-Layouts, die die meist genutzten Diagramm-Strukturen umfassen. Abbildung 4.7 zeigt Beispiele für Diagramme, die einer der Unterklassen der jeweiligen Grundlayouts zugeordnet sind. Die sieben Fluss-Layouts sind Straight, L, Multiline, Stairs, Snake, U und Z. Interessant ist an dieser Stelle ein Vergleich des Detailgrads der hier gezeigten Grundlayouts und der sechs von Lübke und Wutke betrachteten Layout-Richtungen. Letztere nennen Richtungen wie Left-Right und Top-Down, die in der hier genutzten Gruppierung beide zu Straight gehören. Ebenso unterscheiden sie Multiline und Snake jeweils in zwei Varianten (horizontal und vertikal). Es fehlen jedoch die weiteren hier genannten Layouts, wobei unklar ist, ob Diagramme mit diesen Layouts in der manuellen Klassifikation einer anderen Gruppe zugeordnet, oder diese Diagramme als nicht klassifizierbar eingestuft wurden. Denkbar wäre zum Beispiel, dass das Diagramm für U in Abb. 4.7 von Lübke und Wutke als Snake-Vertical klassifiziert werden würde, dies lässt sich jedoch, aufgrund der fehlenden Formalisierung der betrachteten Fluss-Layouts, nicht nachvollziehen.

Die Wahl der Grundlayouts sollte nach der Diskussion von Abb. 4.1 weitestgehend nachvollziehbar sein. Allerdings besteht möglicherweise noch Unklarheit, warum Stairs und Straight nicht dieselbe Kategorie beschreiben. Wenn das Fluss-Layout von einem allgemeinen gerichteten Graphen beschrieben werden soll, wären diese zwei Strukturen abgesehen von der Orientierung nicht zu unterscheiden. BPMN-Diagramme unterscheiden sich in diesem Punkt von allgemeinen Graph-Darstellungen, da sowohl Editoren als auch BPMN an sich eine gewisse Orthogonalität vorschreiben. Zum Beispiel können Symbole für Aktivitäten, anders als bei Darstellungen von allgemeinen Graphen, die meist Kreise als Symbole für die Knoten verwenden, nicht gedreht werden. Auch andere BPMN-Konzepte, wie Pools oder Swimlanes können nur horizontal oder vertikal angeordnet werden.

4.4.2 Untere Hierarchie-Ebenen: Orientierung und Variation

Die oben beschriebene höchste Hierarchiestufe ordnet das Fluss-Layout grob der Struktur entsprechend ein. Die nächste Ebene beschreibt die Orientie-

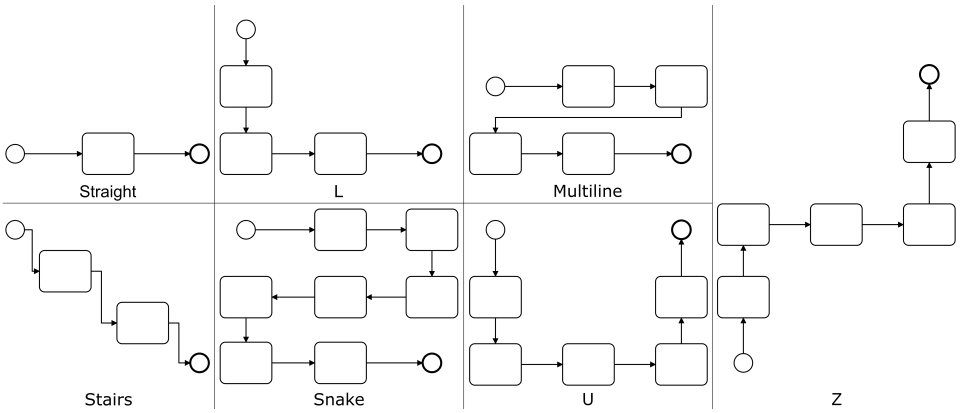

Abbildung 4.7: Die sieben Fluss-Layouts der höchsten Hierarchiestufe

rung dieser Struktur in dem betrachteten Diagramm. Die Oberkategorie Straight hat ebenso wie Stairs vier Unterkategorien auf dieser Ebene: Straight-N, Straight-E, Straight-S und Straight-W bzw. Stairs-NE, Stairs-SE, Stairs-SW und Stairs-NW. Wobei E (Osten), S (Süden), W (Westen) und N (Norden) für die jeweilige Himmelsrichtung stehen. Folglich ist das Stairs Beispiel aus Abb. 4.7 bspw. genauer mit Stairs-SE zu beschreiben, da es nach unten-rechts (Süd-Osten) gerichtet ist. Die übrigen fünf Oberkategorien weisen aufgrund der geringeren Symmetrie jeweils acht Unterkategorien auf der zweiten Ebene auf. Dies wird durch das Beispiel des Multiline-Layouts in Abb. 4.8 illustriert.

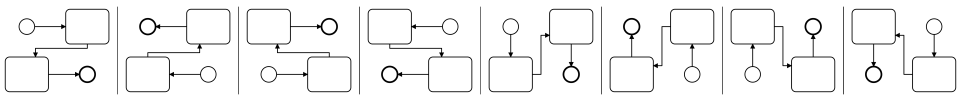

Abbildung 4.8: Die acht Orientierungen für Multiline (von links nach rechts): Multiline-ES, Multiline-WN, Multiline-EN, Multiline-WS, Multiline-SE, Multiline-NW, Multiline-NE und Multiline-SW.

Auf der dritten und finalen Hierarchie-Ebene werden Variationen des Fluss-Layouts unterschieden. Abbildung 4.9 zeigt exemplarisch Beispiele für die betrachteten Varianten des Straight-E Fluss-Layouts. Wie genau die Varianten abgegrenzt werden, wird in Abschnitt 4.6 beschrieben.

4.4.3 Das Straight Grund-Layout

Diagramme, die gerade in eine der vier Himmelsrichtungen verlaufen, haben das Grund-Layout Straight. Dementsprechend gibt es für dieses Grund-Layout vier Orientierungen: E, S, W und N. Da das Straight-Layout das mit Abstand am häufigsten genutzte Grund-Layout ist, können hier sehr viele Varianten identifiziert werden. So werden, wie in Abb. 4.9 gezeigt,

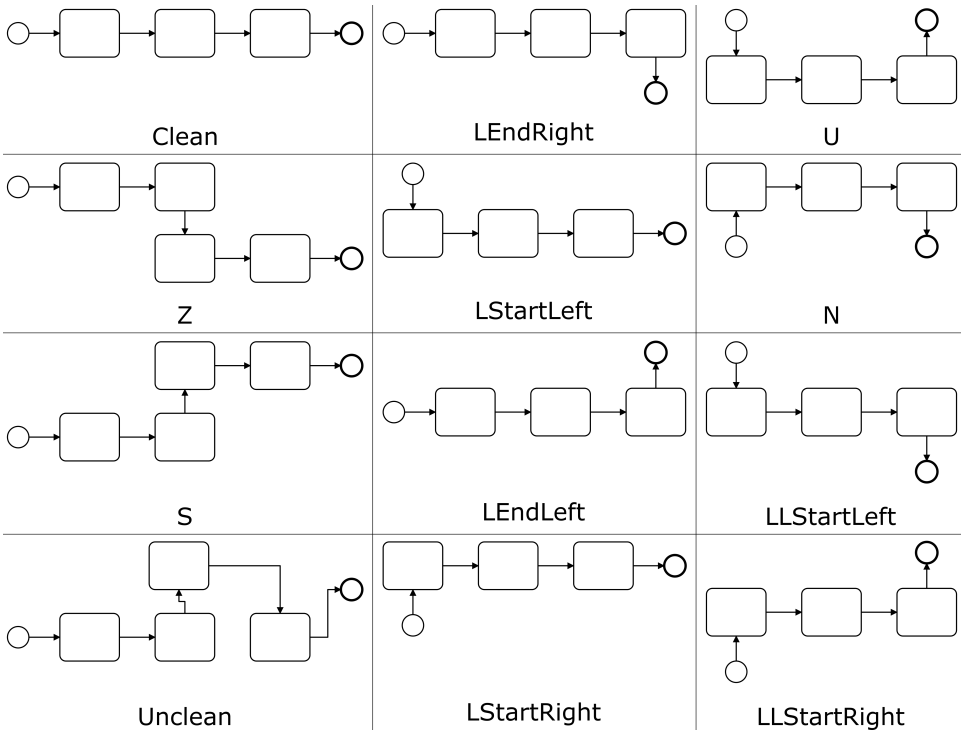

Abbildung 4.9: Betrachtete Varianten des Straight-E Fluss-Layouts

die 12 Varianten Clean, LEndRight, LStartLeft, LEndLeft, LStartRight, LLStartLeft, LLStartRight, U, N, Z und S unterschieden.

Clean bedeutet, dass ein Diagramm nicht maßgeblich von dem Grund-Layout abweicht. Auch hier gibt es Spielräume, da auch Diagramme als Clean bezeichnet werden, bei denen minimale Abweichungen vorkommen. Wie groß diese Abweichungen sein dürfen, wird in Abschnitt 4.6 definiert. Die Variante LEndRight wird für Diagramme gewählt, bei denen fast das gesamte Diagramm Clean ist, nur das End-Objekt maßgeblich nach rechts abweicht. Die Richtung der Abweichung ist bei den Variantenbeschreibungen so zu verstehen, als würde entlang der Fluss-Richtung eine Kurve in die genannte Richtung (hier rechts) gemacht werden. Bei Straight-E-LEndRight weicht das End-Objekt somit nach unten ab, während es bspw. bei Straight-S-LEndRight nach links und bei Straight-W-LEndRight nach oben abweicht. Analog sind die Varianten LStartLeft, LStartRight und LEndLeft zu verstehen. Bei den Varianten LLStartLeft, LLStartRight, U und N werden die oben genannten Abweichungen kombiniert. Für LLStartLeft bedeutet dies, dass sowohl am Start eine Linkskurve als auch am Ende eine Rechtskurve vorhanden ist. Bei Straight-N-LLStartLeft weicht das Start-Objekt nach links und das End-Objekt nach rechts ab. Bei der Variante U wird LStartLeft mit LEndLeft kombiniert und bei Variante N (gespiegeltes U) LStartRight

und LEndRight. Anders ist es bei den Varianten Z und S, da hier die Abweichungen nicht am Start oder am Ende des Diagramms liegen, sondern an einer beliebigen Stelle in der Mitte des Diagramms zu finden sind. Die Variante Z liegt vor, falls die Abweichung nach unten gerichtet ist, wenn das Diagramm so gedreht wurde, dass es nach rechts ausgerichtet ist. S liegt analog dann vor, wenn die Abweichung nach oben gerichtet ist. Für Straight-W-S ist die Abweichung somit nach unten gerichtet. Herauszustellen ist, dass die Abweichung bei allen bisher vorgestellten Varianten nur einen einzelnen Sequenz-Fluss betrifft. Auf diese Weise wird die Trennung der Grund-Layouts gesichert. Das in Abb. 4.7 dargestellte Beispieldiagramm für L wäre dementsprechend nicht mehr L, sondern Straight-S-LEndLeft, wenn eines der beiden auf dem Fluss am Ende liegenden Objekte entfernt werden würde. An dieser Stelle wird klar, welches Fluss-Layout für das in der Einleitung dieser Arbeit gezeigt Diagramm (siehe Abb. 1.1) vorliegt. Da nur der letzte Sequenz-Fluss nach Osten von der sonst gerade nach Süden gerichteten Struktur abweicht hat liegt das Fluss-Layout Straight-S-EndLeft vor. Die letzte betrachtete Variante ist Unclean. Hierzu werden alle Diagramme gezählt, die zwar in eine Richtung ausgerichtet sind, aber einfach oder mehrfach deutlich vom Grund-Layout abweichen.

4.4.4 Komplexe Grund-Layouts

Alle BPMN-Diagramme könnten ein gerades Layout haben. Zwar werden in der Praxis zwar ein Großteil der Modelle auf diese Weise dargestellt, doch ist dieses Prinzip aus unterschiedlichen Gründen nicht universell anzuwenden. Zum Beispiel können mit Layouts wie Snake oder Multiline Modelle mit langen Pfaden auf einem, üblicherweise nicht sehr schmalen, Bildschirm, Blatt Papier, oder einer Tafel dargestellt werden. Außerdem können durch Layouts wie Stairs Metaphern vermittelt werden. Betrachter assoziieren mit dem Stairs-SE Layout Abläufe ähnlich dem Wasserfallmodell, sodass es einen inhaltlichen Mehrwert bietet.

Wie oben dargelegt hängt die Anzahl der möglichen Orientierungen eines Layouts von dessen Symmetrie ab. Während es deshalb nur die vier genannten Straight- und Stairs-Orientierungen gibt, weisen alle anderen Grund-Layouts acht Orientierungen auf. Die ersten zwei Richtungen des Layouts werden konsistent mit einer Abfolge von Himmelsrichtung bezeichnet. Die acht Orientierungen dieser fünf weniger symmetrischen Layouts lauten SE, WS, NW, EN, SW, WN, NE und ES. Hier steht z. B. „SE“ also nicht für Süd-Osten sondern für Süden-Osten, da ein solches Layout zunächst nach Süden und dann nach Osten gerichtet ist.

Die betrachteten Varianten unterscheiden sich je nach Grund-Layout, denn zum einen ist die Formalisierung der Varianten mit großem Aufwand verbunden, weshalb die genaue Beschreibung möglicher Varianten von weniger genutzten Layouts wie U oder Z nicht priorisiert wird. Zum

anderen lassen sich die Straight-Varianten nicht auf alle Layouts übertragen. Snake-ES-EndRight würde zum Beispiel keinen Sinn ergeben, da dies keine Abweichung vom Snake-ES Layout darstellt. Für diese Arbeit wurden 10 Varianten für Z und L definiert, die den Straight-Varianten, allerdings ohne die Varianten Z und S entsprechen. Für Multiline werden die vier Varianten Clean, StartToSide, EndToSide und StartAndEndToSide betrachtet. Aufgrund fehlender Spiegelsymmetrie ist es nicht möglich, die Bezeichnungen der Straight-Varianten zu übernehmen. Während für Snake lediglich Clean und StartToSide unterschieden werden, wurden für U und Stairs gar keine Varianten definiert.

4.5 Erzwungene Abweichungen vom Fluss-Layout

Zwei Konzepte der BPMN erzwingen Abweichungen des Diagramms vom Grund-Layout. Zum einen werden bei Verzweigungen durch Splits, Joins oder Boundary-Events Richtungsänderungen nötig, da ansonsten mehrere Pfade übereinander gezeichnet werden müssten. Zum anderen erzeugen Swimlane-Wechsel in der Regel Richtungswechsel, da innerhalb einer Swimlane meist, je nach Ausrichtung des Pools, horizontal bzw. vertikal gelayoutet wird. Deshalb muss beim Swimlane-Wechsel ein zu der Grundrichtung orthogonaler Versatz eingefügt werden. Diese Abweichungen sollten bei der Bestimmung des Fluss-Layouts eines Diagramms in den meisten Fällen nicht berücksichtigt werden, da sie unvermeidbar sind. Hier ist es wichtig, die Motivation hinter der Betrachtung von Fluss-Layouts im Hinterkopf zu behalten. Wenn bspw. überprüft werden soll, ob Diagramme die Guideline Straight-E-Clean gerichtet zu sein, erfüllen, ist es nicht hilfreich, wenn alle Diagramme mit Swimlane-Wechseln oder Verzweigungen diese Guideline nicht erfüllen können. Obwohl der Datensatz sehr viele triviale Diagramme enthält (über 15% der Diagramme enthalten drei oder weniger Fluss-Knoten) kommen in mehr als der Hälfte der Diagramme entweder Swimlane-Wechsel oder Verzweigungen vor. Abbildung 4.10 zeigt ein Diagramm, das streng von links nach rechts gerichtet ist. Dementsprechend weist es das Fluss-Layout Straight-E-Clean auf. Wie oben beschrieben ergeben sich auch in diesem Diagramm durch Verzweigungen und Swimlane-Wechsel erzwungene Abweichungen vom Grundlayout. Diese werden bei der Klassifikation des Diagramms nicht betrachtet. Das dargestellte Diagramm wird in der folgenden formalen Definition mehrfach als Beispieldiagramm genutzt.

Allerdings ist zu beachten, dass in bestimmten Fällen die orthogonal zur Swimlane-Richtung verlaufenden Komponenten der Swimlane-Wechsel für die Orientierung des Fluss-Layouts relevant sind. Dies wird in Abbildung 4.11 demonstriert. Hier wird die Orientierung des Snake- oder Multiline-Layouts auch durch die Frage bestimmt, ob sich der Start-Knoten in der oberen oder in der unteren Swimlane befindet. Wenn die oben beschriebenen

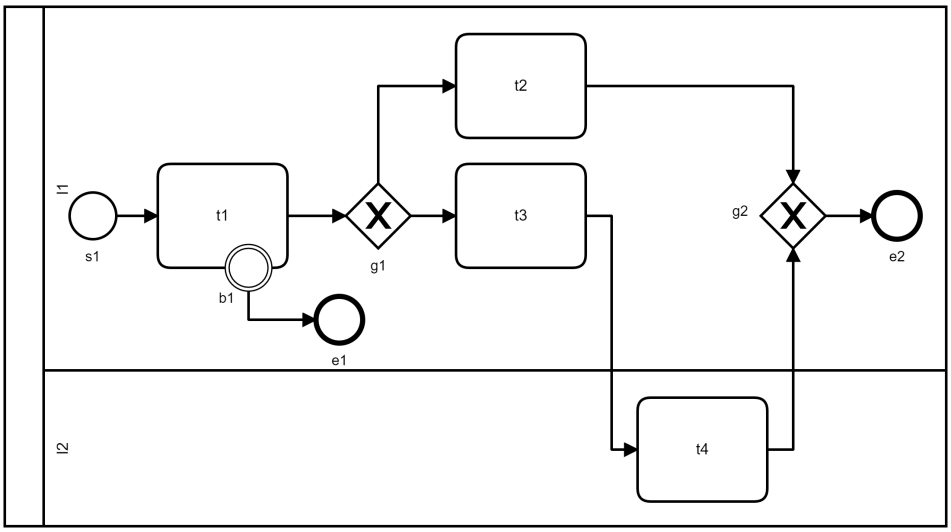

Abbildung 4.10: Straight-E-Clean Diagramm mit Verzweigungen und Swimlane-Wechseln

Abweichungen vom Grundlayout komplett ignoriert werden, könnten zum Beispiel Snake-ES und Snake-EN Layouts nicht voneinander unterschieden werden. Abbildung 4.12 zeigt den gleichen Effekt, der auftritt, wenn Verzweigungen an bestimmten Stellen in einem Diagramm vorkommen. Hier ist die Abweichung von der grundsätzlich horizontalen Richtung nicht nur wegen der Verzweigung, sondern auch wegen des Multiline-Layouts nötig. Ähnlich des oberen Beispiels der Swimlane-Wechseln ist auch hier die vertikale Komponente der Linien-Wechsel für die Orientierung des Multiline-Layouts relevant.

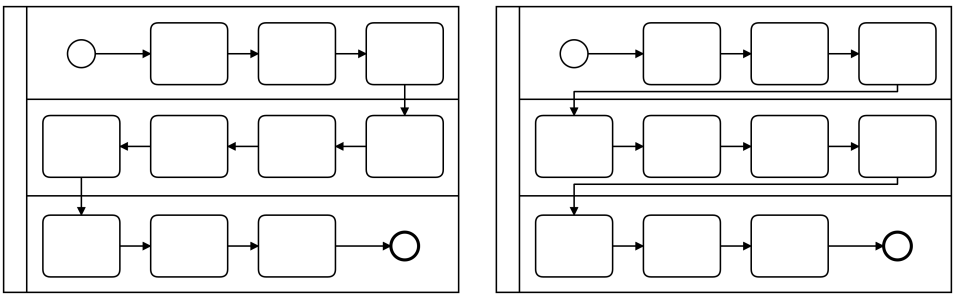

Abbildung 4.11: Links: Snake-ES-Clean, rechts: Multiline-ES-Clean. Diagramme bei denen die Swimlane-Wechsel für die Orientierung des Fluss-Layouts relevant sind.

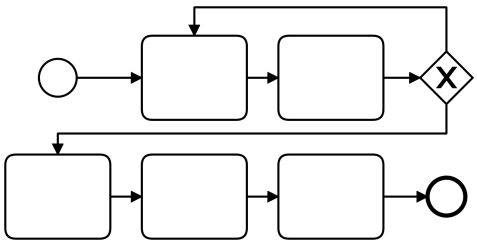

Abbildung 4.12: Multiline-ES-Clean mit Split am Ende der ersten Zeile

4.6 Formale Definition der Layout-Hierarchie

In diesem Abschnitt wird formalisiert, welche Anforderungen an die erweiterbare Layout-Hierarchie gestellt und wie die in dieser Arbeit unterschiedenen Fluss-Layouts definiert werden.

Ein BPMN-Modell ist ein Tupel $M = (V, E, X, Y, B_{\updownarrow}, B_{\leftrightarrow}, L_{\updownarrow}, L_{\leftrightarrow}, l, c)$, wobei $G = (V, E)$ ein gerichteter Graph, V die Menge aller Fluss-Objekte und $E \subseteq V \times V$ die Menge aller Fluss-Verbindungen zwischen Fluss-Objekten ist. Als Fluss-Verbindungen gelten hierbei sowohl Sequenz-Flüsse als auch die Verknüpfungen zwischen Boundary-Ereignissen und den Aktivitäten an die diese angehängt sind. X sei die Menge von Start-Objekten: $X = \{x \in V | d_G^-(x) = 0\}$ und Y die Menge von End-Objekten: $Y = \{y \in V | d_G^+(y) = 0\}$ ($d_G(v)$ bezeichnet den Eingangs- ($^-$) beziehungsweise Ausgangs- ($^+$) Grad des Knoten v im Graphen G). Zudem sei $B_{\updownarrow} \subset V$ die Menge von Boundary-Ereignissen die oben oder unten an eine Aktivität angehängt sind, $B_{\leftrightarrow} \subset V$ die Menge von Boundary-Ereignissen die links oder rechts an eine Aktivität angehängt sind und $L_{\updownarrow}$ die Menge von vertikalen und $L_{\leftrightarrow}$ die Menge von horizontalen Swimlanes sowie $l : V \to L_{\updownarrow} \cup L_{\leftrightarrow}$ eine Abbildung, die ein Fluss-Objekt auf die Swimlane abbildet, in der es sich befindet. Letztendlich ist $c : V \to \mathbb{R} \times \mathbb{R}$ die Abbildung, die ein Fluss-Objekt auf die Koordinaten seines Mittelpunkts abbildet.

Sei $\mathbb{M}$ die Menge aller BPMN-Modelle. Gesucht wird die Funktion $flow : \mathbb{M} \to F$, die jedem BPMN-Modell $M \in \mathbb{M}$ genau ein Fluss-Layout aus der Menge aller Fluss-Layouts F zuordnet. Sei R die Menge aller betrachteten Grund-Layouts, O_r die Menge aller betrachteten Orientierungen für das Grund-Layout $r \in R$ und $V_{r,o}$ die Menge aller betrachteten Varianten für das Grund-Layout $r \in R$ mit der Orientierung $o \in O_g$. Dementsprechend ist die Menge aller betrachteten Fluss-Layouts: $F = \{f = (r, o, v) | r \in R$ und $o \in O_r$ und $v \in V_{r,o}\}$. Alle R, O und V Mengen beinhalten jeweils ein Element, welches den Diagrammen zugewiesen wird, deren Grund-Layout/ Orientierung/ Variante nicht betrachtet wird oder nicht eindeutig festgestellt werden kann. Dieses Element wird im Folgenden mit $Null$ bezeichnet.

Die Funktion $flow$ wird aufgrund ihrer Komplexität in mehrere Funktionen aufgeteilt, die folgend einzeln aufgeführt werden. $flow$ ist definiert

als:

$$flow(M) = (dflow \circ map(pflow, paths(M))$$

Wobei $(f \circ g)(x) = f(g(x))$ die hintereinander Ausführung der Funktionen f und g ist.

4.6.1 Vorbereitung

Das Fluss-Layout eines Diagramms hängt von dem Fluss-Layout der einzelnen Pfade durch das Diagramm ab. Deshalb wird zunächst das Fluss-Layout von Pfaden in einem Diagramm definiert ($pflow$), bevor das gesamte Fluss-Layout abhängig von den Pfad-Layouts definiert wird ($dflow$).

Um dies zu ermöglichen. wird die Hilfsfunktion $map : B^A \times \mathcal{P}(A) \rightarrow \mathcal{P}(B)$ benötigt. A und B sind Mengen und B^A bezeichnet wie üblich die Menge aller Funktionen mit Definitionsmenge A und Bildmenge B. map führt die Funktion aus dem ersten Parameter für jedes Element der Menge aus dem zweiten Parameter aus und gibt die resultierende Menge zurück. Dementsprechend ist sie wie folgt definiert:

$$map((f : A \rightarrow B), C \subseteq A) = \{b \in B | \exists a \in C \text{ mit } f(a) = b\}$$

Um zu verdeutlichen, wie die Funktion map genutzt werden kann wird hier ein Beispiel gegeben: Sei $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}, (a, b) \mapsto a + b$ die Funktion, die zwei natürliche Zahlen addiert. Dann gilt: $map(f, \{(1, 3), (8, 1), (1, 1)\} = \{4, 9, 2\}$.

In der Definition des Fluss-Layouts mit der Funktion $flow$ wird map dafür genutzt, die Funktion $pflow$, die für einen Pfad das Fluss-Layout definiert, auf alle schleifenlosen Pfade von einem Start- zu einem End-Objekt anzuwenden.

Diese Menge der betrachteten Pfade wird durch die Funktion $paths$: $\mathbb{M} \rightarrow \mathcal{P}(P) \times \mathbb{M}$ definiert, wobei $P = \{p = (v_1, \ldots, v_n) | v_i \in V \, \forall i \in \mathbb{N}, i \in [1, n]\}$. V ist an dieser Stelle als „Menge aller Knoten“ zu sehen, da die tatsächliche Knotenmenge je nach BPMN-Modell unterschiedlich ist und deshalb hier in der Definition des Bildbereichs nicht feststeht. P umfasst somit alle Pfade in jedem möglichen Graphen. Die Funktion bestimmt alle schleifenlosen Pfade von einem Start- zu einem End-Objekt in einem BPMN-Modell, und gibt die Menge dieser Pfade jeweils als Tupel zusammen mit dem BPMN-Modell zurück. Die Verknüpfung mit dem BPMN-Modell ist nötig, da Informationen hieraus in später gezeigten Funktionen benötigt werden. Die Funktion ist definiert als:

$$\begin{aligned} paths(M = (V, E, X, Y, B_{\updownarrow}, B_{\leftrightarrow}, L_{\updownarrow}, L_{\leftrightarrow}, l, c)) = \\ \{p = ((v_1, \ldots, v_n), M) | v_i, \in V \, \forall i \in \mathbb{N}, i \in [1, n] \\ \text{und } (v_i, v_{i+1}) \in E \, \forall i \in \mathbb{N}, i \in [1, n-1] \\ \text{und } v_i \neq v_j \, \forall i, j \in \mathbb{N}, i, j \in [1, n] \\ \text{und } v_1 \in X \text{ und } v_n \in Y\} \end{aligned}$$

Hier ist V (wie im Parameter angegeben) die Menge aller Knoten (Fluss-Objekte) in dem Modell M. Um zu verdeutlichen, wie sich die Definition der Fluss-Layouts zusammensetzt, wird das Beispiel-Modell aus Abb. 4.10 verwendet. Es gilt:

$M = (V, E, X, Y, B_{\updownarrow}, B_{\leftrightarrow}, L_{\updownarrow}, L_{\leftrightarrow}, l, c)$

mit $V = \{\text{s1}, \text{t1}, \text{t2}, \text{t3}, \text{t4}, \text{b1}, \text{g1}, \text{g2}, \text{e1}, \text{e2}\}$

und $E = \{(\text{s1}, \text{t1}), (\text{t1}, \text{b1}), (\text{b1}, \text{e1}), (\text{t1}, \text{g1}), (\text{g1}, \text{t2}), (\text{t2}, \text{g2}), (\text{g2}, \text{e2}), (\text{g1}, \text{t3}), (\text{t3}, \text{t4}), (\text{t4}, \text{g2})\}$

und $X = \{\text{s1}\}$

und $Y = \{\text{e1}, \text{e2}\}$

und $B_{\updownarrow} = \{\text{b1}\}$

und $B_{\leftrightarrow} = \{\}$

und $L_{\updownarrow} = \{\}$

und $L_{\leftrightarrow} = \{\text{l1}, \text{l2}\}$

und $l(v) = \begin{cases} \text{l2} & v \in \{\text{t4}\} \\ \text{l1} & \textit{sonst} \end{cases}$

und $c = \{(\text{s1}, (220, 240)), (\text{t1}, (320, 240)), (\text{t2}, (540, 140)), (\text{t3}, (540, 240)), (\text{t4}, (690, 420)), (\text{b1}, (340, 280)), (\text{g1}, (440, 240)), (\text{g2}, (760, 240)), (\text{e1}, (410, 320)), (\text{e2}, (840, 240))\}$ (in Relationsschreibweise)

dementsprechend gilt:

$paths(M) = \{((\text{s1}, \text{t1}, \text{b1}, \text{e1}), M), ((\text{s1}, \text{t1}, \text{g1}, \text{t2}, \text{g2}, \text{e2}), M), ((\text{s1}, \text{t1}, \text{g1}, \text{t3}, \text{t4}, \text{g2}, \text{e2}), M)\}$

Wie gewünscht ergibt die Funktion *paths* die Menge aller für das Fluss-Layout relevanten Pfade, jeweils verknüpft mit dem BPMN-Modell.

4.6.2 Definition des Pfad-Layouts

Die oben vorgestellten Funktionen *map* und *paths* werden in $flow$ genutzt, um die Funktion $pflow$ für alle Pfade aufzurufen. Im Folgenden wird die Funktion $pflow$: betrachtet, die das Fluss-Layout für einzelne Pfade definiert. Da auch diese Funktion zu komplex ist, um sie am Stück darzustellen, wird sie in mehrere Funktionen zerlegt. Es gilt: $pflow = match \circ discr \circ simpl \circ chain$. Die vier hintereinander ausgeführten Funktionen werden in den nächsten Abschnitten einzeln definiert und erläutert.

Definition von Vektorketten

Die zuerst ausgeführte Funktion ist die Funktion $chain : P \times \mathbb{M} \to C$ mit $C = \{(\vec{w}_1, \ldots, \vec{w}_n) | n \in \mathbb{N} \text{ und } \vec{w}_i \in W \, \forall i \in \mathbb{N}, i \in [1, n]\}$ und $W = \{\binom{a}{b} | a, b \in$

$\mathbb{R}\}$. Die Menge C beinhaltet also alle Tupel von zweistelligen Vektoren. Ein Element von C wird Vektorkette genannt. Die Funktion *chain* wandelt einen Layout-Pfad in eine Vektorkette um. Definiert ist sie wie folgt:

$$chain(p = (v_1, \ldots, v_n), M) = (\vec{w}_1, \ldots, \vec{w}_{n-1})$$
$$\text{mit } \vec{w}_i = vect(i, p, M)\, \forall i \in \mathbb{N}, i \in [1, n-1]$$

Jedes Element der resultierenden Vektorkette wird durch die Funktion $vect : \mathbb{N} \times P \times \mathbb{M} \to W$ definiert. Diese Funktion bestimmt den Vektor für eine bestimmte Kante aus einem Pfad. Sie bildet dementsprechend den Index einer Kante, zusammen mit dem Pfad und dem BPMN-Modell auf einen Vektor ab. Hierbei sind die erzwungenen Abweichungen vom Fluss-Layout (siehe Abschnitt 4.5) zu beachten. Dies wird in der Definition von *vect* berücksichtigt, indem je nach erzwungener Abweichung eine, oder beide Komponenten des Ergebnisvektors auf 0 gesetzt werden. Einzelheiten werden in Abschnitt 4.6.4 erklärt. Die Funktion ist durch die folgende Fallunterscheidung definiert:

$$
\begin{aligned}
&vect(i, p = (v_1, \ldots, v_n), M = (V, E, X, Y, B_{\updownarrow}, B_{\leftrightarrow}, L_{\updownarrow}, L_{\leftrightarrow}, l, c)) \\
&= \begin{cases}
\binom{0}{0} & v_{i+1} \in B_{\updownarrow} \cup B_{\leftrightarrow} \\
\binom{0}{y} & \begin{array}{l} v_i \in B_{\leftrightarrow} \\ \text{oder } d^+_G(v_i) > 1 \text{ und } i \neq 1 \text{ und } prevX \leq prevY \\ \text{oder } d^+_G(v_i) > 1 \text{ und } i = 1 \text{ und } outAvgX \leq outAvgY \\ \text{oder } d^-_G(v_{i+1}) > 1 \text{ und } d^+_G(v_{i+1}) > 1 \text{ und } inAvgX \leq inAvgY \\ \text{oder } d^-_G(v_{i+1}) > 1 \text{ und } d^+_G(v_{i+1}) = 1 \text{ und } nextX \leq nextY \\ \text{oder } d^-_G(v_{i+1}) > 1 \text{ und } i + 1 = n \text{ und } inAvgX \leq inAvgY \\ \text{oder } l(v_i) \neq l(v_{i+1}) \text{ und } l(v_i) \in L_{\updownarrow} \end{array} \\
\binom{x}{0} & \begin{array}{l} v_i \in B_{\updownarrow} \\ \text{oder } d^+_G(v_i) > 1 \text{ und } i \neq 1 \text{ und } prevX > prevY \\ \text{oder } d^+_G(v_i) > 1 \text{ und } i = 1 \text{ und } outAvgX > outAvgY \\ \text{oder } d^-_G(v_{i+1}) > 1 \text{ und } d^+_G(v_{i+1}) > 1 \text{ und } inAvgX > inAvgY \\ \text{oder } d^-_G(v_{i+1}) > 1 \text{ und } d^+_G(v_{i+1}) = 1 \text{ und } nextX > nextY \\ \text{oder } d^-_G(v_{i+1}) > 1 \text{ und } i + 1 = n \text{ und } inAvgX > inAvgY \\ \text{oder } l(v_i) \neq l(v_{i+1}) \text{ und } l(v_i) \in L_{\leftrightarrow} \end{array} \\
\binom{x}{y} & \text{sonst}
\end{cases} \\
&\text{mit } G = (V, E) \\
&\text{und } x = x_2 - x_1 \text{ und } y = y_2 - y_1 \text{ und } (x_1, y_1) = c(v_i) \\
&\text{und } (x_2, y_2) = c(v_{i+1}) \text{ und } \tbinom{prevX}{prevY} = vect(i - 1, p, M) \\
&\text{und } \tbinom{outAvgX}{outAvgY} = avgDirection \circ map(\\
&\qquad edgeToVect, \{e = ((a, b), M) | (a, b) \in E \text{ und } a = v_i\}) \\
&\text{und } \tbinom{inAvgX}{inAvgY} = avgDirection \circ map(\\
&\qquad edgeToVect, \{e = ((a, b), M) | (a, b) \in E \text{ und } b = v_{i+1}\}) \\
&\text{und } \tbinom{nextX}{nextY} = avgDirection \circ map(\\
&\qquad edgeToVect, \{e = ((a, b), M) | (a, b) \in E \text{ und } a = v_{i+1}\})
\end{aligned}
$$

wobei die Funktion $edgeToVect : E \times \mathbb{M} \to W$ eine Kante auf einen Vektor abbildet, der den Mittelpunkt der Knoten die durch die Kante verbunden werden verbindet. Sie ist definiert als:

$$
\begin{aligned}
&edgeToVect(e = (v_1, v_2), M = (V, E, X, Y, B_{\updownarrow}, B_{\leftrightarrow}, L_{\updownarrow}, L_{\leftrightarrow}, l, c)) \\
&\quad = \tbinom{x}{y} \text{ mit } x = x_2 - x_1 \text{ und } y = y_2 - y_1 \\
&\qquad \text{und } (x_1, y_1) = c(v_1) \text{ und } (x_2, y_2) = c(v_2)
\end{aligned}
$$

Außerdem bestimmt die Funktion $avgDirection : \mathcal{P}(W) \to \mathbb{R}$ die durchschnittliche Richtung einer Menge von Vektoren. Sie ist wie folgt definiert:

$$avgDirection(W = \{\vec{w}_1, \ldots, \vec{w}_n\}) = arctan2\left(\frac{\frac{\vec{w}_1}{|\vec{w}_1|} + \cdots + \frac{\vec{w}_n}{|\vec{w}_n|}}{n}\right)$$

Wie bei den vorherigen Schritten wird auch die Bildung der Vektorketten anhand des Beispieldiagramms verdeutlicht. *chain* wird durch die *map* Funktion für alle drei Pfade (Ergebnis von *path*) aufgerufen. In der Folgenden Rechnung werden zur besseren Verständlichkeit an manchen Stellen die Variablenbezeichnungen aus den Funktionsdefinitionen in Klammern und kleiner Schrift vor die tatsächlichen Werte geschrieben.

$p_1 = (\mathrm{s1}, \mathrm{t1}, \mathrm{b1}, \mathrm{e1})$
$chain(p_1, M) = (vect(1, p_1, M), vect(2, p_1, M), vect(3, p_1, M))$
$= \left(\binom{100}{0}, \binom{0}{0}, \binom{70}{0}\right)$
denn:
$vect(1, p_1, M) = \binom{320-220}{240-240} = \binom{100}{0}$
$vect(2, p_1, M) = \binom{0}{0}$ da $_{(v_{i+1} =)}\mathrm{b1} \in B_{\updownarrow}$
$vect(3, p_1, M) = \binom{410-340}{0} = \binom{70}{0}$ da $_{(v_i =)}\mathrm{b1} \in B_{\updownarrow}$

$p_2 = (\mathrm{s1}, \mathrm{t1}, \mathrm{g1}, \mathrm{t2}, \mathrm{g2}, \mathrm{e2})$
$chain(p_2, M) = (vect(1, p_2, M), vect(2, p_2, M), vect(3, p_2, M),$
$vect(4, p_2, M), vect(5, p_2, M))$
$= \left(\binom{100}{0}, \binom{100}{0}, \binom{100}{0}, \binom{220}{0}, \binom{80}{0}\right)$
denn:
$vect(1, p_2, M) = \binom{320-220}{240-240} = \binom{100}{0}$
$vect(2, p_2, M) = \binom{420-320}{240-240} = \binom{100}{0}$
$vect(3, p_2, M) = \binom{540-420}{0} = \binom{100}{0}$ da $d_G^+(_{(v_i =)}\mathrm{g1}) > 1$
und $_{(i =)}3 \neq 1$ und $_{(prevX =)}100 >$ $_{(prevY =)}0$
$vect(4, p_2, M) = \binom{760-540}{0} = \binom{220}{0}$ da $d_G^-(_{(v_{i+1} =)}\mathrm{g2}) > 1$
und $d_G^+(_{(v_{i+1} =)}\mathrm{g2}) = 1$ und $_{(nextX =)}80 >$ $_{(nextY =)}0$
$vect(5, p_2, M) = \binom{840-760}{240-240} = \binom{80}{0}$

$$p_3 = (\mathrm{s1}, \mathrm{t1}, \mathrm{g1}, \mathrm{t3}, \mathrm{t4}, \mathrm{g2}, \mathrm{e2})$$
$$\begin{aligned} chain(p_3, M) &= (vect(1, p_3, M), vect(2, p_3, M), vect(3, p_3, M), \\ &\quad vect(4, p_3, M), vect(5, p_3, M), vect(6, p_3, M)) \\ &= (\tbinom{100}{0}, \tbinom{100}{0}, \tbinom{100}{0}, \tbinom{150}{0}, \tbinom{70}{0}, \tbinom{80}{0}) \end{aligned}$$

denn:

$vect(1, p_3, M) = \binom{320-220}{240-240} = \binom{100}{0}$

$vect(2, p_3, M) = \binom{420-320}{240-240} = \binom{100}{0}$

$vect(3, p_3, M) = \binom{540-420}{0} = \binom{100}{0}$ da $d_G^+({}_{(v_i =)}\mathrm{g1}) > 1$
und ${}_{(i =)}3 \neq 1$ und ${}_{(prevX =)}100 > {}_{(prevY =)}0$

$vect(4, p_3, M) = \binom{690-540}{0} = \binom{150}{0}$
da ${}_{(l(v_i) =)}\mathrm{l1} \neq {}_{(l(v_{i+1}) =)}\mathrm{l2}$ und $\mathrm{l1} \in L_{\leftrightarrow}$

$vect(5, p_3, M) = \binom{760-690}{0} = \binom{70}{0}$
da ${}_{(l(v_i) =)}\mathrm{l2} \neq {}_{(l(v_{i+1}) =)}\mathrm{l1}$ und $\mathrm{l2} \in L_{\leftrightarrow}$

$vect(6, p_3, M) = \binom{840-760}{240-240} = \binom{80}{0}$

Vereinfachung von Vektorketten

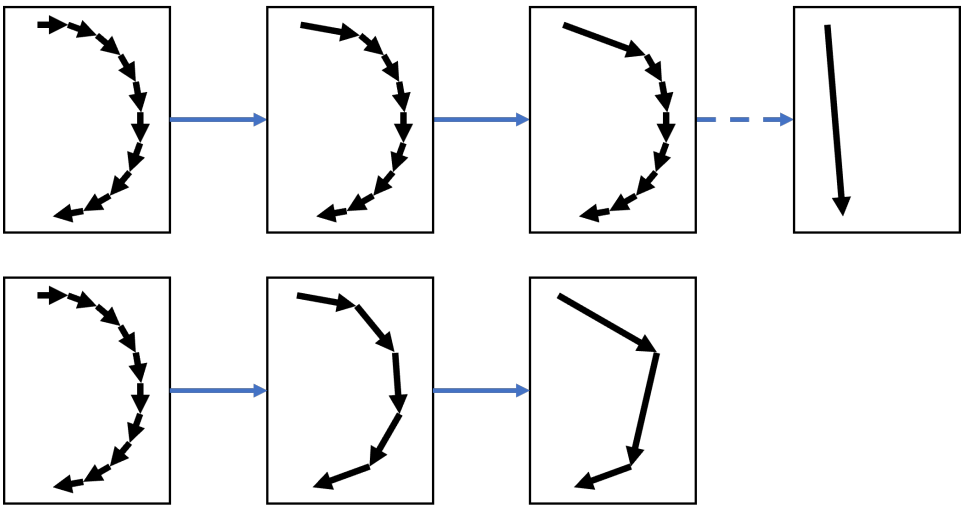

Abbildung 4.13: Oben: Kombination von Vektoren ähnlicher Richtung von vorne nach hinten. Unten: Rekursive paarweise Kombination von Vektoren ähnlicher Richtung.

In der Funktion $pflow$ wird nach der Funktion *chain* die Funktion $simpl : C \to C \times \mathcal{P}(W)$ ausgeführt, wobei C die Menge aller Vektorketten und W die Menge aller zweistelligen Vektoren wie in der Definition von *chain* angegeben

ist. Die Funktion *simpl* vereinfacht eine Vektorkette, indem rekursiv immer wieder aufeinanderfolgende Vektoren, die eine ähnliche Richtung haben, kombiniert werden, bis keine Vereinfachung mehr möglich ist. Abbildung 4.13 zeigt einen Vergleich zwischen der hier gewählten rekursiven Variante (unten) und einer anderen Möglichkeit, bei der die Vektorkette von vorne nach hinten vereinfacht wird (oben). Der Vorteil der rekursiven Variante besteht darin, dass die grobe Form der Vektorkette bestehen bleibt, was für die Definition von Fluss-Layouts essenziell ist. Die Funktion *simpl* ist mit mehreren Hilfsfunktionen wie folgt definiert:

$$\begin{aligned}&simpl(c = (\vec{w}_1, \ldots, \vec{w}_n))\\ &= (combine(c),\\ &\quad \{\vec{v}_i | (\vec{v}_1, \ldots, \vec{v}_m) = combine(c) \text{ und } i \notin notCombined(c, \{1, \ldots, n\})\})\end{aligned}$$

Wobei die Funktion $combine : C \to C$ die Vereinfachung vornimmt, und die Funktion $notCombined : C \times \mathcal{P}(\mathbb{N}) \to \mathcal{P}(\mathbb{N})$ die Indizes der Vektoren sammelt, die nicht mit anderen aus der Vektorkette kombiniert wurden. *simpl* gibt als zweiten Rückgabewert die Menge der kombinierten Vektoren aus der vereinfachten Vektorkette zurück. Diese Unterscheidung zwischen kombinierten und nicht kombinierten Vektoren wird in späteren Funktionen benötigt. Die beiden Funktionen sind wie folgt definiert ($|c|$ steht für die Anzahl der Vektoren in der Vektorkette c):

$$combine(c = (\vec{w}_1, \ldots, \vec{w}_n)) = \begin{cases} c & |c| = |s(c)| \\ combine(s(c)) & sonst \end{cases}$$

$$notCombined(c = (\vec{w}_1, \ldots, \vec{w}_n), I) = \begin{cases} I' & |c| = |s(c)| \\ notCombined(s(c), I') & sonst \end{cases}$$

$$\begin{aligned}\text{mit } I' = \{&k = i - |\{a \in \mathbb{N}, a \in [1, i-1] | close(\vec{w}_a, \vec{w}_{a+1}) = 1\\ &\text{und } close(\vec{w}_{a-1}, \vec{w}_a) \neq 1\}| \; | i \in I\\ &\text{und } |s((\vec{w}_1, \ldots, \vec{w}_{i-1}))| \neq |s((\vec{w}_1, \ldots, \vec{w}_i))| \neq |s((\vec{w}_1, \ldots, \vec{w}_{i+1}))|\}\end{aligned}$$

Die Funktion $s : C \to C$ definiert einen einzelnen Rekursionsschritt.

$$s(c = (\vec{w}_1, \ldots, \vec{w}_n)) = \begin{cases} (\vec{w}_1) & n = 1 \\ (\vec{w}_1 + \vec{w}_2) & n = 2 \text{ und } close(\vec{w}_1, \vec{w}_2) = 1 \\ (\vec{w}_1, \vec{w}_2) & n = 2 \text{ und } close(\vec{w}_1, \vec{w}_2) \neq 1 \\ s((\vec{w}_1, \ldots, \vec{w}_{n-2})) \cdot (\vec{w}_{n-1} + \vec{w}_n) & \begin{array}{l} n > 2 \text{ und } |s((\vec{w}_1, \ldots, \vec{w}_{n-2}))| \\ \neq |s((\vec{w}_1, \ldots, \vec{w}_{n-1}))| \\ \text{und } close(\vec{w}_{n-1}, \vec{w}_n) = 1 \end{array} \\ s((\vec{w}_1, \ldots, \vec{w}_{n-1})) \cdot (\vec{w}_n) & \text{sonst} \end{cases}$$

wobei $\cdot$ die Konkatenation zweier Vektorketten ist

Die Funktion $close : W^2 \to \{0, 1\}$ gibt an, ob der kleinere Winkel zwischen zwei Vektoren kleiner als $\frac{360^\circ}{16}$ ist.

An dieser Stelle wird erneut das Beispielmodell betrachtet. Allerdings wird darauf verzichtet, die Vereinfachung für alle Pfade detailliert aufzuführen, da die Pfade ähnlich behandelt werden. Repräsentativ wird der zweite Pfad (s1,t1,g1,t2,g2,e2) betrachtet.

$simpl\left(c = \left(\binom{100}{0}, \binom{100}{0}, \binom{100}{0}, \binom{220}{0}, \binom{80}{0}\right)\right) = \left(\left(\binom{600}{0}\right), \left\{\binom{600}{0}\right\}\right)$
denn:
$combine(c) = s(s(s(c))) = \left(\binom{600}{0}\right)$
drei Rekursionsschritte, denn $|s(s(c))| = 1 = |s(s(s(c)))|$

1. Rekursionsschritt:
$s(c) = \left(\binom{200}{0}, \binom{320}{0}, \binom{80}{0}\right)$
Rechenweg:
$s\left(\binom{100}{0}\right) = \left(\binom{100}{0}\right)$
$s\left(\binom{100}{0}, \binom{100}{0}\right) = \left(\binom{200}{0}\right)$
$s\left(\binom{100}{0}, \binom{100}{0}, \binom{100}{0}\right) = \left(\binom{200}{0}\right) \cdot \left(\binom{100}{0}\right) = \left(\binom{200}{0}, \binom{100}{0}\right)$
$s\left(\binom{100}{0}, \binom{100}{0}, \binom{100}{0}, \binom{220}{0}\right) = \left(\binom{200}{0}\right) \cdot \left(\binom{320}{0}\right) = \left(\binom{200}{0}, \binom{320}{0}\right)$
$s\left(\binom{100}{0}, \binom{100}{0}, \binom{100}{0}, \binom{220}{0}, \binom{80}{0}\right) = \left(\binom{200}{0}, \binom{320}{0}\right) \cdot \left(\binom{80}{0}\right)$
$= \left(\binom{200}{0}, \binom{320}{0}, \binom{80}{0}\right)$

2. Rekursionsschritt:
$s(s(c)) = \left(\binom{520}{0}, \binom{80}{0}\right)$
Rechenweg:
$s\left(\binom{200}{0}\right) = \left(\binom{200}{0}\right)$
$s\left(\binom{200}{0}, \binom{320}{0}\right) = \left(\binom{520}{0}\right)$
$s\left(\binom{200}{0}, \binom{320}{0}, \binom{80}{0}\right) = \left(\binom{520}{0}\right) \cdot \left(\binom{80}{0}\right) = \left(\binom{520}{0}, \binom{80}{0}\right)$

3. Rekursionsschritt:
$s(s(s(c))) = s\left(\binom{520}{0}, \binom{80}{0}\right) = \left(\binom{600}{0}\right)$

Da alle Vektoren der Vektorkette die gleiche Richtung haben, gilt $close(v_1, v_2) = 1$ für alle Kombinationen von Vektoren. Aus diesem Grund gilt $notCombined(c) = \{\}$. Dies ist für alle Vektorketten des Beispieldia-

gramms gleich. Das Ergebnis von *simpl* für alle drei Vektorketten ist:

$$simpl\left(\binom{100}{0},\binom{0}{0},\binom{70}{0}\right) = \left(\left(\binom{170}{0}\right),\{\binom{170}{0}\}\right)$$
$$simpl\left(\left(\binom{100}{0},\binom{100}{0},\binom{100}{0},\binom{220}{0},\binom{80}{0}\right)\right) = \left(\left(\binom{600}{0}\right),\{\binom{600}{0}\}\right)$$
$$simpl\left(\binom{100}{0},\binom{100}{0},\binom{100}{0},\binom{150}{0},\binom{70}{0},\binom{80}{0}\right) = \left(\left(\binom{600}{0}\right),\{\binom{600}{0}\}\right)$$

Diskretisierung der Vektorrichtungen

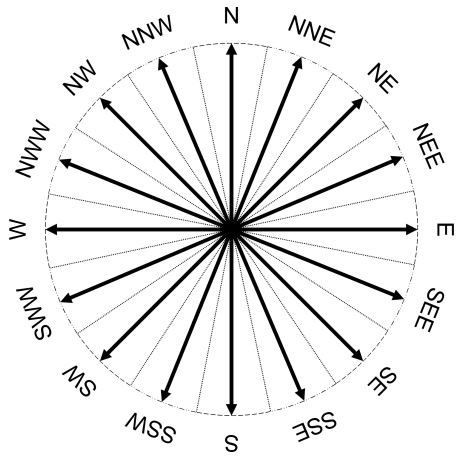

Abbildung 4.14: Diskrete Vektor-Richtungen

Nach der Funktion *simpl* wird in $flow$ die Funtkion $discr : C \times \mathcal{P}(W) \to \Sigma^*$ aufgerufen, die die Richtungen der Vektoren in der vereinfachten Vektorkette diskretisiert. Σ ist die Menge aller betrachteten diskreten Vektor-Richtungen in den vier Varianten markiert-lang, nicht-markiert-lang, markiert-kurz und nicht-markiert-kurz. Markiert bedeutet, dass der Vektor, dessen Richtung diskretisiert wurde, eine Kombination aus mehreren Vektoren aus der ursprünglichen nicht vereinfachten Vektorkette ist. Diese Markierung wird in einer späteren Funktion, genauso wie die Unterscheidung zwischen lang (mehr als halb so lang im Vergleich zum längsten Vektor in der Vektorkette) und kurz, zur Definition des Fluss-Layouts benötigt.

Wie in Abb. 4.14 gezeigt werden 16 Vektor-Richtungen unterschieden. Σ beinhaltet für jede der 16 Richtungen vier Varianten. Es wird zwischen markierten (x-**M**-y) und nicht markierten (x-**NM**-y), sowie langen (x-y-**L**) und kurzen (x-y-**S**) Vektoren unterschieden. Dementsprechend gilt $\Sigma = \{$ E-M-L, E-M-S, E-NM-L, E-NM-S, SSE-M-L, SSE-M-S, SSE-NM-L, SSE-NM-S,$\ldots\}$. *discr* ist wie folgt definiert:

$discr(c = (\vec{w}_1, \ldots, \vec{w}_n), Z) = \alpha\beta\gamma\ldots$
mit $\alpha, \beta, \gamma, \cdots \in \Sigma$ und $\alpha = dis(\vec{w}_1, A, Z), \beta = dis(\vec{w}_2, A, Z), \ldots$
wobei $A = \{\vec{v} = \vec{w}_i | i \in \mathbb{N}, i \in [1, n]\}$
und $dis : W \times \mathcal{P}(W) \times \mathcal{P}(W) \to \Sigma$
einen Vektor diskretisiert:
$dis(\vec{w} = \binom{a}{b}, A, Z) =$

$$
\begin{cases}
\boxed{\text{W-M-L}} & \frac{d \cdot 16}{\pi} \in [, -15] \text{ und } \vec{w} \in Z \text{ und } |\vec{w}| > \frac{m}{2} \\
\boxed{\text{W-M-S}} & \frac{d \cdot 16}{\pi} \in [, -15] \text{ und } \vec{w} \in Z \text{ und } |\vec{w}| \leq \frac{m}{2} \\
\boxed{\text{W-NM-L}} & \frac{d \cdot 16}{\pi} \in [, -15] \text{ und } \vec{w} \notin Z \text{ und } |\vec{w}| > \frac{m}{2} \\
\boxed{\text{W-NM-S}} & \frac{d \cdot 16}{\pi} \in [, -15] \text{ und } \vec{w} \notin Z \text{ und } |\vec{w}| \leq \frac{m}{2} \\
 & \\
\boxed{\text{NWW-M-L}} & \frac{d \cdot 16}{\pi} \in [-15, -13] \text{ und } \vec{w} \in Z \text{ und } |\vec{w}| > \frac{m}{2} \\
\boxed{\text{NWW-M-S}} & \frac{d \cdot 16}{\pi} \in [-15, -13] \text{ und } \vec{w} \in Z \text{ und } |\vec{w}| \leq \frac{m}{2} \\
\boxed{\text{NWW-NM-L}} & \frac{d \cdot 16}{\pi} \in [-15, -13] \text{ und } \vec{w} \notin Z \text{ und } |\vec{w}| > \frac{m}{2} \\
\boxed{\text{NWW-NM-S}} & \frac{d \cdot 16}{\pi} \in [-15, -13] \text{ und } \vec{w} \notin Z \text{ und } |\vec{w}| \leq \frac{m}{2} \\
 & \\
\vdots & \vdots \\
 & \\
\boxed{\text{E-M-L}} & \frac{d \cdot 16}{\pi} \in [-1, 1] \text{ und } \vec{w} \in Z \text{ und } |\vec{w}| > \frac{m}{2} \\
\boxed{\text{E-M-S}} & \frac{d \cdot 16}{\pi} \in [-1, 1] \text{ und } \vec{w} \in Z \text{ und } |\vec{w}| \leq \frac{m}{2} \\
\boxed{\text{E-NM-L}} & \frac{d \cdot 16}{\pi} \in [-1, 1] \text{ und } \vec{w} \notin Z \text{ und } |\vec{w}| > \frac{m}{2} \\
\boxed{\text{E-NM-S}} & \frac{d \cdot 16}{\pi} \in [-1, 1] \text{ und } \vec{w} \notin Z \text{ und } |\vec{w}| \leq \frac{m}{2} \\
 & \\
\vdots & \vdots \\
 & \\
\boxed{\text{SWW-M-L}} & \frac{d \cdot 16}{\pi} \in [13, 15] \cdot \frac{\pi}{16} \text{ und } \vec{w} \in Z \text{ und } |\vec{w}| > \frac{m}{2} \\
\boxed{\text{SWW-M-S}} & \frac{d \cdot 16}{\pi} \in [13, 15] \cdot \frac{\pi}{16} \text{ und } \vec{w} \in Z \text{ und } |\vec{w}| \leq \frac{m}{2} \\
\boxed{\text{SWW-NM-L}} & \frac{d \cdot 16}{\pi} \in [13, 15] \cdot \frac{\pi}{16} \text{ und } \vec{w} \notin Z \text{ und } |\vec{w}| > \frac{m}{2} \\
\boxed{\text{SWW-NM-S}} & \frac{d \cdot 16}{\pi} \in [13, 15] \cdot \frac{\pi}{16} \text{ und } \vec{w} \notin Z \text{ und } |\vec{w}| \leq \frac{m}{2} \\
 & \\
\boxed{\text{W-M-L}} & \frac{d \cdot 16}{\pi} \in [15,] \cdot \frac{\pi}{16} \text{ und } \vec{w} \in Z \text{ und } |\vec{w}| > \frac{m}{2} \\
\boxed{\text{W-M-S}} & \frac{d \cdot 16}{\pi} \in [15,] \cdot \frac{\pi}{16} \text{ und } \vec{w} \in Z \text{ und } |\vec{w}| \leq \frac{m}{2} \\
\boxed{\text{W-NM-L}} & \frac{d \cdot 16}{\pi} \in [15,] \cdot \frac{\pi}{16} \text{ und } \vec{w} \notin Z \text{ und } |\vec{w}| > \frac{m}{2} \\
\boxed{\text{W-NM-S}} & \frac{d \cdot 16}{\pi} \in [15,] \cdot \frac{\pi}{16} \text{ und } \vec{w} \notin Z \text{ und } |\vec{w}| \leq \frac{m}{2}
\end{cases}
$$

mit $d = arctan2(b, a)$
und $m = \dfrac{\max\limits_{\vec{v} \in A} |\vec{v}|}{2}$

Für das Beispiel-Modell bedeutet das:

$$discr\left(\left(\binom{170}{0}\right), \{\binom{170}{0}\}\right) = dis\left(\binom{170}{0}, \{\binom{170}{0}\}, \{\binom{170}{0}\}\right) = \boxed{\text{E-M-L}}$$

$$\text{denn } \frac{arctan2(170,0)\cdot 16}{\pi} = 0 \in [-1,1] \text{ und } \left|\binom{170}{0}\right| > \frac{\left|\binom{170}{0}\right|}{2}$$

analog für die anderen beiden Vektorketten:

$$discr\left(\left(\binom{600}{0}\right), \{\binom{600}{0}\}\right) = \boxed{\text{E-M-L}}$$

$$discr\left(\left(\binom{600}{0}\right), \{\binom{600}{0}\}\right) = \boxed{\text{E-M-L}}$$

Reguläre Ausdrücke

Die letzte Funktion, die für alle Pfade ausgeführt wird, ist die Funktion $match : \Sigma^* \to F$, die der diskretisierten vereinfachten Vektorkette ein Fluss-Layout zuweist. Die Formalisierung geschieht an dieser Stelle über reguläre Ausdrücke. Während diese hier exemplarisch angegeben werden, findet sich in Anhang A die vollständige Liste.

Um die Lesbarkeit der regulären Ausdrücke zu gewährleisten, werden einige Abkürzungen definiert. Die Abkürzungen sind Konkatenationen von einzelnen Symbolen aus Σ. So gilt: $\boxed{\text{N-M}}$:=$\boxed{\text{N-M-L}}\boxed{\text{N-M-S}}$ sowie $\boxed{\text{N-NM}}$:=$\boxed{\text{N-NM-L}}\boxed{\text{N-NM-S}}$ und $\boxed{\text{N-L}}$:=$\boxed{\text{N-M-L}}\boxed{\text{N-NM-L}}$ als auch $\boxed{\text{N-S}}$:=$\boxed{\text{N-M-S}}\boxed{\text{N-NM-S}}$ und letztendlich $\boxed{\text{N}}$:=$\boxed{\text{N-M}}\boxed{\text{N-NM}}$. Diese Abkürzungen sind analog für alle anderen 15 Richtungen definiert. Als Syntax für die regulären Ausdrücke wird die von Java Patterns verwendet [30]. $[\boxed{\text{SE-M}}]$ bezeichnet bspw. den Ausdruck, der das Symbol $\boxed{\text{SE-M-S}}$ oder $\boxed{\text{SE-M-L}}$ akzeptiert. Folglich akzeptiert der Ausdruck Vektoren, die nach Süd-Osten gerichtet und markiert sind. Die definierten Abkürzungen enthalten keine eckigen Klammern, damit innerhalb von eckigen Klammern mehrere Abkürzungen verwendet werden können. So akzeptiert bspw. $[\boxed{\text{S}}\boxed{\text{E}}]$ alle nach Süden oder Osten gerichteten Vektoren, unabhängig von Länge und Markierung. Zusätzlich zu den oben gezeigten werden weitere Abkürzungen eingeführt. Diese werden ebenfalls in Anhang A aufgezählt.

Die Funktion *match* ist wie folgt definiert:

$$match(s) = \begin{cases} (Straight, E, Clean) & matches(s, [\boxed{\text{E}}]) = 1 \\ (Straight, W, Z) & matches(s, [\boxed{\text{W-M}}][\boxed{\text{NWW-TO-N-NM}}][\boxed{\text{W-M}}]) = 1 \\ \vdots & \vdots \\ (\text{Null}, \text{Null}, \text{Null}) & \text{sonst} \end{cases}$$

Die Funktion *matches* ist dabei genau dann 1, wenn der reguläre Ausdruck aus dem zweiten Parameter die Eingabe aus dem ersten Parameter akzeptiert.

Für das Beispiel-Modell ist das matching sehr einfach, da jede diskretisierte vereinfachte Vektorkette nur aus einer Richtung besteht. Es gilt

$matches(\boxed{\text{E-M-L}}, [\boxed{\text{E}}]) = matches(\boxed{\text{E-M-L}}, [\boxed{\text{E-M-L}}, \boxed{\text{E-M-S}}, \boxed{\text{E-NM-L}}, \boxed{\text{E-NM-S}}]) = 1$ und somit $match(\boxed{\text{E}}) = (Straight, E, Clean)$.

4.6.3 Definition des Diagramm-Layouts

Nachdem definiert wurde, welches Fluss-Layout ein Pfad hat, wird nun durch die Funktion $dflow : \mathcal{P}(F) \to F$ das Fluss-Layout gesamter BPMN-Modellen definiert, indem die Menge von Pfad-Layouts auf ein Diagramm-Layout abgebildet werden.

$$dflow(P = \{(r_1, o_1, v_1), (r_2, o_2, v_2), \ldots, (r_n, o_n, v_n)\}) = (r, o, v)$$

$$\text{mit } r = \begin{cases} \underset{x \in R}{\arg\max}\ countR(P, x) & \max\limits_{x \in R} countR(P, x) \geq \frac{2}{3} \cdot |P| \\ \text{Null} & sonst \end{cases}$$

wobei $countR : \mathcal{P}(F) \times R \to \mathbb{N}$ wie folgt definiert ist:

$$countR(P = \{(r_1, o_1, v_1), (r_2, o_2, v_2), \ldots, (r_n, o_n, v_n)\}, x)$$
$$= |\{l = (a, b, c) | a = x\}|$$

$$\text{und } o = \begin{cases} \underset{x \in O_r}{\arg\max}\ countO(P, x) & r \neq \text{Null und } \max\limits_{x \in O_r} countO(P, x) \geq \frac{2}{3} \cdot |P| \\ \text{Null} & sonst \end{cases}$$

wobei $countO : \mathcal{P}(F) \times O_r \to \mathbb{N}$ wie folgt definiert ist:

$$countO(P = \{(r_1, o_1, v_1), (r_2, o_2, v_2), \ldots, (r_n, o_n, v_n)\}, x)$$
$$= |\{l = (a, b, c) | a = r \text{ und } b = x\}|$$

$$\text{und } v = \begin{cases} \underset{x \in V_{r,o}}{\arg\max}\ countV(P, x) & r, o \neq \text{Null u. } \max\limits_{x \in V_{r,o}} countV(P, x) \geq \frac{2}{3} \cdot |P| \\ \text{Null} & sonst \end{cases}$$

wobei $countV : \mathcal{P}(F) \times V_{r,o} \to \mathbb{N}$ wie folgt definiert ist:

$$countV(P = \{(r_1, o_1, v_1), (r_2, o_2, v_2), \ldots, (r_n, o_n, v_n)\}, x)$$
$$= |\{l = (a, b, c) | a = r \text{ und } b = o \text{ und } c = x\}|$$

Dementsprechend hat das Beispieldiagramm das Fluss-Layout $dflow(\{(Straight, E, Clean), (Straight, E, Clean), (Straight, E, Clean)\}) = (Straight, E, Clean)$.

4.6.4 Bestimmung angepasster Vektoren

Wie oben erläutert, bestehen Vektorketten aus Vektoren, die zwischen den Mittelpunkten zweier durch Sequenz-Flüsse oder Parent-Boundary-Ereignis-Verknüpfungen verbundener Fluss-Objekte aufgespannt werden. Allerdings werden die Vektoren aufgrund der in Abschnitt 4.5 vorgestellten erzwungenen Abweichungen von der Grundrichtung in bestimmten Fällen angepasst. Hierzu sind in Abb. 4.15 Beispiele gezeigt, anhand derer das grundlegende Prinzip erklärt wird. Es ist jeweils ein Fluss-Objekt (v_1)

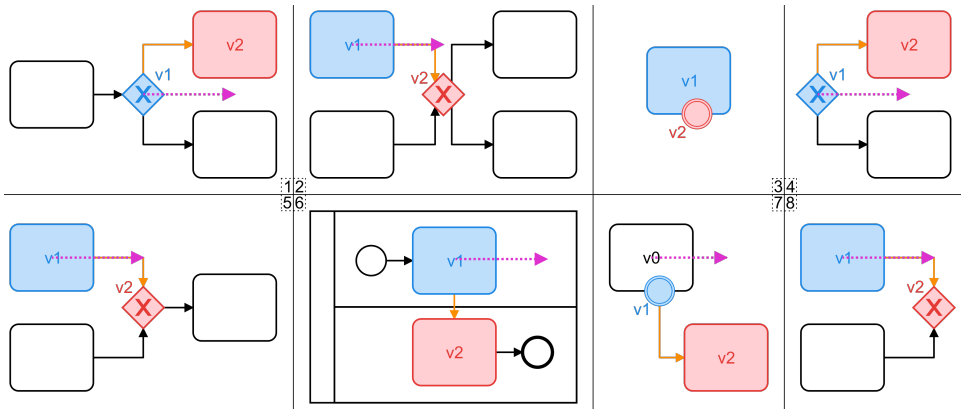

Abbildung 4.15: Beispieldiagramme für Fälle in denen der Vektor zwischen den blau und den rot dargestellten Fluss-Objekt angepasst werden muss. Der resultierende Vektor ist gestrichelt in Violett eingezeichnet.

blau und ein Fluss-Objekt (v_2) rot gefärbt. Der Vektor zwischen den beiden wird betrachtet. Die gestrichelten, violetten Vektoren sind die tatsächlichen Vektoren, die in die Vektorkette eingehen. Die Position dieser Vektoren ist in der Abbildung so gewählt, wie sie in der Vektorkette positioniert wären, wenn diese beim Mittelpunkt des Start-Objektes für den Pfad beginnen würde. Die folgende Tabelle zeigt, wie in den jeweiligen Fällen vorgegangen wird. Meist wird eine der beiden Komponenten des Vektors auf 0 gesetzt. Um dies kompakt darstellen zu können wird die horizontale Komponente x und die vertikale Komponente y genannt.

Fall	Fallbeschreibung	Umgang mit dem Fall
1	v_1 hat mehrere ausgehende und mindestens einen eingehenden Sequenz-Fluss	Wenn ein Vektor, der in die durchschnittliche Richtung der eingehenden Sequenz-Flüsse von v_1 zeigt, eher horizontal ist ($\|x\| > \|y\|$), wird y auf 0 gesetzt, sonst x
2	v_2 hat mehrere eingehende und mehrere ausgehende Sequenz-Flüsse	Wenn ein Vektor, der in die durchschnittliche Richtung der eingehenden Sequenz-Flüsse von v_2 zeigt, eher horizontal ist, wird y auf 0 gesetzt, sonst x
3	v_2 ist als Boundary-Ereignis an v_1 angehängt	Sowohl x als auch y werden auf 0 gesetzt
4	v_1 hat mehrere ausgehende Sequenz-Flüsse und keinen eingehenden Sequenz-Fluss	Wenn ein Vektor, der in die durchschnittliche Richtung der ausgehenden Sequenz-Flüsse zeigt, eher horizontal ist, wird y auf 0 gesetzt, sonst x
5	v_1 hat mehrere ausgehende Sequenz-Flüsse und genau einen eingehenden Sequenz-Fluss	Wenn der, in der Vektorkette vorherige Vektor eher horizontal ist wird y auf 0 gesetzt, sonst x
6	v_1 ist in einer anderen Swimlane als v_2	Wenn die Lanes horizontal sind, wird y auf 0 gesetzt, sonst x
7	v_1 ist als Boundary-Ereignis an v_0 angehängt	Wenn v_1 an der oberen oder unteren Kante von v_0 angehängt ist wird y auf 0 gesetzt, sonst x
8	v_2 hat mehrere eingehende und keinen ausgehenden Sequenz-Fluss	Wenn ein Vektor, der in die durchschnittliche Richtung der eingehenden Sequenz-Flüsse von v_2 zeigt, eher horizontal ist, wird y auf 0 gesetzt, sonst x

Kapitel 5

Automatisierte Klassifikation des Fluss-Layouts

Aufbauend auf den formalisierten Fluss-Layouts wird ein Algorithmus konstruiert und ein Werkzeug implementiert, welches das Layout von BPMN-Diagrammen automatisiert klassifiziert.

5.1 Algorithmus und Implementierung

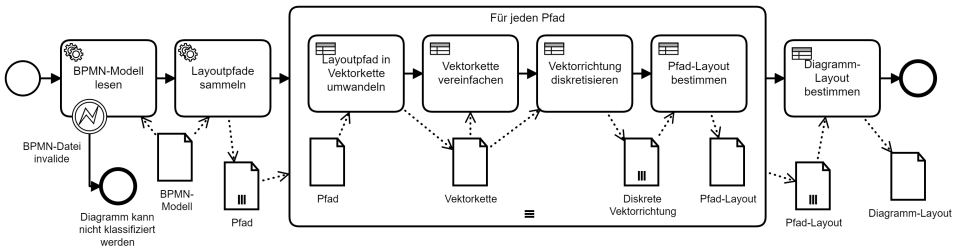

Abbildung 5.1: Algorithmus zur automatisierten Klassifikation des Fluss-Layouts von BPMN-Diagrammen

Der Algorithmus ist, mit einer ähnlichen Struktur wie die Funktionen der Formalisierung, modular aufgebaut. Abbildung 5.1 zeigt jene Schritte, die der Algorithmus durchläuft, um ein BPMN-Modell zu klassifizieren. In diesem Abschnitt wird gezeigt, wie die Anforderungen der Formalisierung algorithmisch gelöst und implementiert werden. Die Implementierung wurde in Java vorgenommen. Die nachfolgend gezeigten Code-Ausschnitte sind teilweise vereinfacht, um die wichtigsten Konzepte herauszustellen, ohne sich in Details zu verlieren. So werden bspw. Klassenzugehörigkeiten der Methoden nur gezeigt, wenn sie für das Verständnis wichtig sind. Methoden, die in den Beispielen auf der ersten Einrückungsebene definiert sind,

könnten unabhängig von Klassen statisch definiert werden. Die vollständige Implementierung ist auf GitHub zu finden [2].

5.1.1 BPMN-Modell lesen

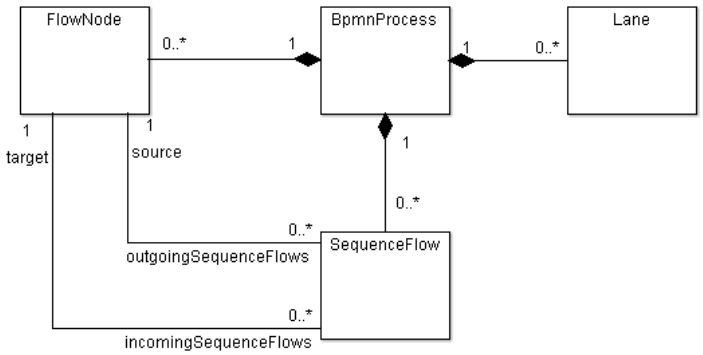

Abbildung 5.2: Deserialisiertes BPMN-Modell, UML-Klassendiagramm (vereinfacht)

Im ersten Schritt wird das BPMN-Modell geparsed. Dies geschieht durch die Implementierung von Daniel Lübkes BPMN-Layout-Analyzer [23]. Außerdem werden einige Metriken ausgewertet, um die Analysierbarkeit des Modells sicherzustellen. Dies wird in Abschnitt 5.2 genauer erläutert. Das UML-Klassendiagramm in Abb. 5.2 zeigt, wie das geparste BPMN-Modell nach diesem ersten Schritt vorliegt. Die für das Fluss-Layout relevanten Elemente eines BPMN-Modells (*BpmnProcess*) sind die Fluss-Objekte (*FlowNode*), Sequenz-Flüsse (*SequenceFlow*) und die Swimlanes (*Lane*). Jeder Sequenz-Fluss verknüpft zwei Fluss-Objekte und jedes Boundary-Ereignis ist an genau ein Fluss-Objekt angebunden. Hierbei können mehrere Boundary-Ereignisse an ein Fluss-Objekt gebunden sein.

5.1.2 Layoutpfade sammeln

Das geparste Modell wird nun weiter analysiert. Zunächst werden alle (schleifenfreien) Pfade (im Code *Traces* genannt) von jedem Start-Objekt zu jedem End-Objekt mittels Tiefensuche bestimmt werden. Die Tiefensuche wird auf dem aus Fluss-Objekten als Knoten und Sequenz-Flüssen bzw. Boundary-Ereignis-Verknüpfungen als Kanten aufgespannten Graphen durchgeführt. Die Start-Objekte (*startFlowNodes*) sind die Fluss-Objekte des BPMN-Modells, die weder eingehende Sequenz-Flüsse haben, noch Boundary-Ereignisse sind. Die Methode *getNonLoopingTracesToEnd* bestimmt rekursiv alle Pfade ausgehend von einem Element und sammelt diese in der übergebenen Liste.

```
List<Trace> traces = new ArrayList<>();
for (FlowNode startNode : startFlowNodes) {
  Trace currentTrace = new Trace();
  currentTrace.addFlowNodeToTrace(fn);
  final List<Trace> result = new ArrayList<>();
  getNonLoopingTracesToEnd(currentTrace, result);
  traces.addAll(result);
}

void getNonLoopingTracesToEnd(Trace currentTrace, List<Trace> traces) {
  FlowNode current = currentTrace.last();
  for(FlowNode following : current.getOutgoingSequenceFlows().stream()
      .map(SequenceFlow::getTarget).collect(Collectors.toList())) {
    if(!currentTrace.contains(following)) {
      Trace newTrace = new Trace(currentTrace);
      newTrace.addFlowNodeToTrace(following);
      getNonLoopingTracesToEnd(newTrace, traces);
    }
  }

  for(FlowNode boundaryEvent : current.getBoundaryEvents()) {
    if(!currentTrace.contains(boundaryEvent)) {
      Trace newTrace = new Trace(currentTrace);
      newTrace.addFlowNodeToTrace(boundaryEvent);
      getNonLoopingTracesToEnd(newTrace, traces);
    }
  }

  if(current.getOutgoingSequenceFlows().size() == 0 &&
     current.getBoundaryEvents().size() == 0) {
    traces.add(currentTrace);
  }
}
```

5.1.3 Layoutpfad in Vektorkette umwandeln

Um anschließend die Fluss-Layouts der Pfade zu bestimmen, werden diese zunächst in Vektorketten umgewandelt. Hierfür wird für jeden Sequenz-Fluss auf jedem Pfad ein Vektor bestimmt. Die Implementierung orientiert sich stark an der Formalisierung. Exemplarisch werden folgend die Fälle 1 und 4 aus Abschnitt 4.6.4 gezeigt. Es wird der Sequenzfluss zwischen den zwei Fluss-Objekten *prevNode* und *thisNode* betrachtet. Zunächst wird der Vektor zwischen den beiden Mittelpunkten (v) bestimmt. Der dritte Parameter des Vector-Kontruktors gibt an, ob der Vektor markiert ist. Dies ist nicht der Fall, da später nur solche Vektoren markiert werden sollen, die aus mehreren Vektoren zusammengesetzt wurden. Die Methode *setCenterPosition* wird zur Speicherung der Vektorposition im Diagramm verwendet. Wenn Fall 1 bzw. Fall 4 vorliegt, hat *prevNode* mehrere ausgehende Sequenz-Flüsse. Der Vektor *prevFlowVector* ist jender, der für den vorherigen Sequenz-Fluss auf dem betrachteten Pfad bestimmt wurde. Dieser ist *null*, wenn *prevNode* ein Start-Objekt ist. Liegt Fall 1 vor, ist *prevFlowVector* nicht *null*. In diesem Fall wird, je nach Richtung von *prevFlowVector*, die entsprechende

Komponente des Vektors v auf 0 gesetzt. Liegt allerdings Fall 4 vor ist dies nicht möglich. Deshalb wird, der Formalisierung entsprechend, die durchschnittliche Richtung der von *prevNode* ausgehenden Sequenz-Flüsse als Grundlage für die Entscheidung, welche Komponente auf 0 gesetzt werden soll, herangezogen. Um zu bestimmen, ob die durchschnittliche Richtung, die im Bogenmaß vorliegt, eher horizontal oder vertikal ist, wird ein Vektor konstruiert, der in die entsprechende Richtung zeigt.

```
Vector v = new Vector(prevNode.getCenter(), thisNode.getCenter(), false);
v.setCenterPosition(add(prevNode.getCenter(),v.divide(2.0)));
if (prevNode.getOutgoingSequenceFlows().size() > 1) {
  if (prevFlowVector != null) {
    setDimensionToZero(prevFlowVector.isHorizontal(), v);
  } else {
    double averageDirection = averageDirection(
        sequenceFlowsToVectors(prevNode.getOutgoingSequenceFlows()));
    setDimensionToZero(isHorizontal(averageDirection), v);
  }
}

Vector add(Vector v1, Vector v2) {
  Vector v = new Vector(v1.getX() + v2.getX(), v1.getY() + v2.getY(), true);
  Vector center1 = v1.getCenterPosition();
  Vector center2 = v2.getCenterPosition();
  Vector toAdd = new Vector(center1, center2).divide(2.0);
  v.setCenterPosition(center1.getX() + toAdd.getX(), center1.getY() + toAdd.getY());
  return v;
}

void setDimensionToZero(boolean dimension, Vector v) {
  if (dimension) {
    v.setY(0);
  } else {
    v.setX(0);
  }
}

List<Vector> sequenceFlowsToVectors(List<SequenceFlow> sequenceFlows) {
  return sequenceFlows.stream().map(sequenceFlow -> {
        FlowNode source = sequenceFlow.getSource();
        FlowNode target = sequenceFlow.getTarget();
        return new Vector(source.getCenter(), target.getCenter());
      }).collect(Collectors.toList());
}

double averageDirection(List<Vector> vectors) {
  return vectors.stream().mapToDouble(Vector::getDirection)
      .average().orElse(Double.NaN);
}

private boolean isHorizontal(double direction) {
  return new Vector(Math.cos(direction), Math.sin(direction)).isHorizontal();
}

public class Vector {
  // ...
  public Vector(double x, double y) {
    this(x, y, false);
  }
```

```
52
53    public Vector(double x, double y, boolean mark) {
54      this.x = x;
55      this.y = y;
56      this.mark = mark;
57    }
58
59    public Vector(Vector v1, Vector v2, boolean mark) {
60      this(v2.getX() - v1.getX(), v2.getY() - v1.getY(), mark);
61    }
62
63    public boolean isHorizontal() {
64      return Math.abs(x) > Math.abs(y);
65    }
66
67    public double getDirection() {
68      return Math.atan2(y, x);
69    }
70
71    public Vector divide(double divisor) {
72      return new Vector(x / divisor, y / divisor);
73    }
74    // ...
75  }
```

5.1.4 Vektorkette vereinfachen

Die Vektorketten werden im nächsten Schritt vereinfacht, indem aufeinanderfolgende Vektoren mit ähnliche Richtung zusammengefasst werden. Der folgende Programmcode zeigt, wie die hierfür verantwortliche Methode *simplifyVectors*, implementiert ist. Der *angleThreshold* ist als Maximalwinkel definiert, ab dem Vektoren kombiniert werden sollen. Die Formalisierung fordert hier den konstanten Wert: $angleThreshold = \frac{2\pi}{16}$. Eine Vektorkette ist eine Liste von Vektoren. Hier wird dementsprechend die Vektorkette *vectors* vereinfacht. Zu beachten ist, dass die Methode *add* (siehe Abschnitt 5.1.3), die zwei Vektoren addiert, einen markierten Vektor zurückgibt. Auf diese Weise werden nur Vektoren markiert, die aus mehreren zusammengesetzt wurden. Außerdem wird in *add* der Mittelpunkt des kombinierten Vektors bestimmt.

```
1  List<Vector> simplifyVectors(List<Vector> vectors, double angleThreshold) {
2    if (vectors.size() == 1) return vectors;
3    List<Vector> simplified = new ArrayList<>();
4    boolean simplifiable = false;
5    for (int i = 0; i < vectors.size() - 1; i++) {
6      Vector v1 = vectors.get(i);
7      Vector v2 = vectors.get(i + 1);
8      double angle = smallestPositiveAngle(v1, v2);
9      if (angle < angleThreshold) {
10       simplified.add(add(v1, v2));
11       simplifiable = true;
12       i++;
13     } else {
14       simplified.add(v1);
15     }
```

```
16      if (i + 1 == vectors.size() - 1) {
17        simplified.add(vectors.get(vectors.size() - 1));
18      }
19    }
20    if (simplifiable) {
21      return simplifyVectors(simplified, angleThreshold);
22    } else {
23      return simplified;
24    }
25  }
26
27  double smallestPositiveAngle(Vector v1, Vector v2) {
28    double absAngle = Math.abs(v1.getDirection() - v2.getDirection());
29    return Math.min(absAngle, 2 * Math.PI - absAngle);
30  }
31
32  public class Vector {
33    // ...
34    public double getLength() {
35      return Math.sqrt(x * x + y * y);
36    }
37    // ...
38  }
```

5.1.5 Vektorrichtung diskretisieren

Nachdem die Vektorketten vereinfacht wurden, werden die Richtungen der Vektoren diskretisiert. Hierfür werden die in der Formalisierung etablierten Konstanten genutzt. Diese weisen als Werte willkürlich gewählte Buchstaben auf, um die im nächsten Schritt erfolgende Implementierung der regulären Ausdrücke, zu vereinfachen. Dementsprechend wird eine Vektorkette diskretisiert als *String* dargestellt.

```
1   String discretizeVectorDirections(List<Vector> vectors) {
2     double maxlength =
        vectors.stream().map(Vector::getLength).max(Double::compare).get();
3     return vectors.stream()
4       .map(v -> discreteDirection(v, v.getLength() >= maxlength / 2))
5       .reduce(String::concat).get();
6   }
7   String discreteDirection(Vector v, boolean isLong) {
8     double pis = Math.PI / 16;
9     double d = v.getDirection();
10    boolean m = v.isMarked();
11    boolean l = isLong;
12    if (d < -15 * pis) return m ? l ? W_M_L   : W_M_S   : l ? W_NM_L   : W_NM_S;
13    if (d < -13 * pis) return m ? l ? NWW_M_L : NWW_M_S : l ? NWW_NM_L : NWW_NM_S;
14    if (d < -11 * pis) return m ? l ? NW_M_L  : NW_M_S  : l ? NW_NM_L  : NW_NM_S;
15    if (d < -19 * pis) return m ? l ? NNW_M_L : NNW_M_S : l ? NNW_NM_L : NNW_NM_S;
16    if (d < -7  * pis) return m ? l ? N_M_L   : N_M_S   : l ? N_NM_L   : N_NM_S;
17    if (d < -5  * pis) return m ? l ? NNE_M_L : NNE_M_S : l ? NNE_NM_L : NNE_NM_S;
18    if (d < -3  * pis) return m ? l ? NE_M_L  : NE_M_S  : l ? NE_NM_L  : NE_NM_S;
19    if (d < -1  * pis) return m ? l ? NEE_M_L : NEE_M_S : l ? NEE_NM_L : NEE_NM_S;
20    if (d < 1   * pis) return m ? l ? E_M_L   : E_M_S   : l ? E_NM_L   : E_NM_S;
21    if (d < 3   * pis) return m ? l ? SEE_M_L : SEE_M_S : l ? SEE_NM_L : SEE_NM_S;
22    if (d < 5   * pis) return m ? l ? SE_M_L  : SE_M_S  : l ? SE_NM_L  : SE_NM_S;
23    if (d < 7   * pis) return m ? l ? SSE_M_L : SSE_M_S : l ? SSE_NM_L : SSE_NM_S;
```

```
24    if (d < 9   * pis) return m ? l ? S_M_L   : S_M_S   : l ? S_NM_L   : S_NM_S;
25    if (d < 11  * pis) return m ? l ? SSW_M_L : SSW_M_S : l ? SSW_NM_L : SSW_NM_S;
26    if (d < 13  * pis) return m ? l ? SW_M_L  : SW_M_S  : l ? SW_NM_L  : SW_NM_S;
27    if (d < 15  * pis) return m ? l ? SWW_M_L : SWW_M_S : l ? SWW_NM_L : SWW_NM_S;
28    else         return m ? l ? W_M_L   : W_M_S   : l ? W_NM_L   : W_NM_S;
29  }
30  public class Constants {
31    public static final String E_M_L = "R";
32    public static final String E_NM_L = "r";
33    public static final String E_M_S = "W";
34    public static final String E_NM_S = "w";
35
36    // ...
37
38    public static final String NEE_M_L = "A";
39    public static final String NEE_NM_L = "a";
40    public static final String NEE_M_S = "B";
41    public static final String NEE_NM_S = "b";
42  }
```

5.1.6 Pfad-Layout bestimmen

Um das Fluss-Layout der einzelnen Pfade zu bestimmen, wird geprüft, ob der jeweilige *String* in einer der, von den in Anhang A gezeigten regulären Ausdrücken, beschriebenen Sprachen enthalten ist. Sollte dies zutreffen, wird das Fluss-Layout für den Pfad entsprechend gesetzt. Die regulären Ausdrücke liegen als Werte mit der jeweiligen Fluss-Layout-Bezeichnung als Schlüssel in der *Map REGEXES*.

```
1  List<String> matchDirections(String pathDirections) {
2    ArrayList<String> possibleLayouts
3    for (Map.Entry<String, String> entry : REGEXES.entrySet()) {
4      if (pathDirections.matches(entry.getValue())) {
5        possibleLayouts.add(entry.getKey());
6      }
7    }
8    return possibleLayouts;
9  }
```

Allerdings ist zu beachten, dass bei Diagrammen, wie dem aus Abb. 4.12 an dieser Stelle mehrere *possibleLayouts* vorliegen können, da die regulären Ausdrücke in diesen Fällen die Orientierung von Multiline- oder Snake-Layouts nicht eindeutig klassifizieren können. Aus diesem Grund wird die diskretisierte Vektorkette angepasst, falls mehrere *possibleLayouts* vorliegen. Hierfür werden die Vektorpositionen in dem Diagramm betrachtet. Das folgende Code-Beispiel zeigt gekürzt, wie diese Anpassung im Falle von Multiline durchgeführt wird:

```
1  String multilineReplacement(String pathDirections, VectorChain vectorChain) {
2    StringBuilder result = new StringBuilder("" + pathDirections.charAt(0));
3    for (int i = 1; i < pathDirections.length(); i++) {
4      String character = "" + pathDirections.charAt(i);
5      Vector centerOfVector = vectorChain.getVectors().get(i).getCenterPosition();
6      Vector centerOfPrevVector = vectorChain.getVectors()
```

```
7           .get(i - 1).getCenterPosition();
8
9       switch (character) {
10        case N_NM_L:
11          if (centerOfVector.getX() > centerOfPrevVector.getX()) {
12            result.append(NNE_NM_L);
13          } else {
14            result.append(NNW_NM_L);
15          }
16          break;
17        case E_NM_L:
18          if (centerOfVector.getY() > centerOfPrevVector.getY()) {
19            result.append(SEE_NM_L);
20          } else {
21            result.append(NEE_NM_L);
22          }
23          break;
24
25        // ...
26
27        default:
28          result.append(character);
29          break;
30      }
31    }
32    return result.toString();
33  }
```

Nachdem die diskretisierte Vektorkette angepasst wurde, wird erneut das matching durchgeführt, da nun auch in den Spezialfällen die Orientierung eindeutig festgestellt werden kann.

5.1.7 Diagramm-Layout bestimmen

Abschließend werden die Fluss-Layouts der einzelnen Pfade (*flowLayoutPerPath*) kombiniert, um das Diagramm-Layout zu ermitteln. Da die Fluss-Layouts als *Strings* vorliegen werden die Methoden *getBaselayoutAndOrientation* und *getBaselayout* genutzt um die einzelnen Hierarchiestufen zu trennen. Wie in der Formalisierung gefordert wird das Fluss-Layout, welches bis zur tiefstmöglichen Hierarchiestufe für mehr als $\frac{2}{3}$ der Pfade identifiziert wurde ausgegeben. Falls dies für keines der Layouts zutrifft wird „*Other*“ als Fluss-Layout genannt um zu verdeutlichen, dass keine Klassifikation möglich war.

```
1   void calculateFlowLayout() {
2     if (flowLayoutPerPath.stream().filter(direction ->
3         direction.equals("Other")).count() > flowLayoutPerPath.size() * (2. / 3.)) {
4       flowLayout = "Other";
5     } else {
6       Map<String, Long> layoutOccurences = flowLayoutPerPath.stream()
7           .collect(Collectors.groupingBy(Function.identity(),Collectors.counting()));
8       int numberOfPaths = flowLayoutPerPath.size();
9       if (!setDominatingLayout(layoutOccurences, numberOfPaths)) {
10        layoutOccurences = flowLayoutPerPath.stream().collect(Collectors.groupingBy(
11            this::getBaselayoutAndOrientation,Collectors.counting()));
12        if (!setDominatingLayout(layoutOccurences, numberOfPaths)) {
```

```
13          layoutOccurences = flowLayoutPerPath.stream().collect(Collectors.groupingBy(
14              this::getBaselayout, Collectors.counting()));
15          if (!setDominatingLayout(layoutOccurences, numberOfPaths)) {
16            flowLayout = "Other";
17          }
18        }
19      }
20    }
21  }
22
23  String getBaselayoutAndOrientation(String flowLayout) {
24    return flowLayout.substring(0,
25        flowLayout.indexOf("-", flowLayout.indexOf("-") + 1) != -1
26            ? flowLayout.indexOf("-", flowLayout.indexOf("-") + 1)
27            : flowLayout.length());
28  }
29
30  String getBaselayout(String flowLayout){
31    return flowLayout.substring(0, flowLayout.contains("-")
32        ? flowLayout.indexOf("-")
33        : flowLayout.length());
34  }
```

5.2 Nicht analysierbare Modelle

Es gibt mehrere Gründe, warum ein BPMN-Modell nicht klassifiziert werden kann. Um einschätzen zu können, wie oft es vorkommt, dass ein Diagramm nicht von dem vorgestellten Werkzeug analysiert werden kann, werden auch hier die jeweiligen Werte für den großen GitHub Datensatz (siehe Abschnitt 4.2) genannt. So ist der am häufigsten auftretende Grund sind invalide XML-Dateien. Dies ist z. B. der Fall, wenn ein Sequenz-Fluss zwei Fluss-Elemente verbindet, aber mindestens eines nicht definiert ist. Diese Art von Fehler tritt für 13438 (ca. 20%) der 67169 BPMN-Modelle auf.

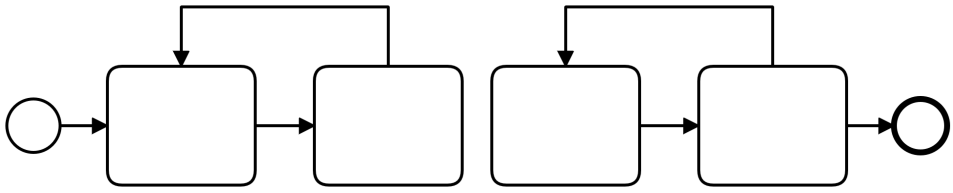

Abbildung 5.3: Zweigeteiltes BPMN-Diagramm, links ohne End-Objekt und rechts ohne Start-Objekt.

Abgesehen von Fehlern, die direkt beim Parsen des Modells auftreten, gibt es auch einige Fehler, die speziell die Analyse des Fluss-Layouts verhindern. Da die dem Werkzeug zu Grunde liegende Formalisierung auf die Existenz von Pfaden zwischen Start- und End-Objekten basiert, müssen diese zwangsläufig vorliegen, damit ein Modell analysiert werden kann. Abbildung 5.3 zeigt ein Diagramm, dass zwar ein Start- und ein End-Objekt

vorweist, jedoch kein Pfad existiert, der die beiden verbindet. Somit ist das Diagramm nicht klassifizierbar. Analysemöglichkeiten solcher Diagramme könnten durch eine Anpassung der Formalisierung gefunden werden, da dies jedoch den Rahmen dieser Arbeit sprengen würde wird an dieser Stelle davon abgesehen. 397 (ca. 0,6%) der Modelle aus dem Datensatz haben kein Start-Objekt, 53 (ca. 0,1%) haben ein Start- aber kein End-Objekt und 35 (ca. 0,05%) haben sowohl ein Start- als auch ein End-Objekt aber das End-Objekt ist vom Start-Objekt nicht erreichbar.

Eine weitere Anforderung, die das Werkzeug an Modelle stellt, ist die Existenz von mindestens einem Sequenz-Fluss. Im GitHub Datensatz gibt es mit 1177 (ca. 1,8%) relativ viele Diagramme, die diese Anforderung nicht erfüllen, da sie keine Sequenz-Flüsse enthalten. Dies sind z. B. Diagramme, in denen Objekte der BPMN lediglich gesammelt werden oder solche Diagramme, die lediglich ein Fluss-Objekt enthalten. Diese Diagramme auf ihr Fluss-Layout zu untersuchen ist unmöglich, da sie keinen Fluss darstellen.

Darüber hinaus werden zwar 712 (ca. 1%) der Modelle im Datensatz durch ein valides XML-Dokument repräsentiert, jedoch fehlen bei diesen Diagrammen Layout-Informationen für die Elemente. Da die Layout-Daten für die Layout-Analyse essenziell sind, werden auch diese Modelle nicht klassifiziert. Abschließend gibt es BPMN-Editoren, die Layout-Informationen nicht standardmäßig speichern. Deshalb wird vor der Klassifikation überprüft, ob die Koordinaten der Sequenz-Flüsse tatsächlich zu den Koordinaten der verbundenen Objekte passen. Für 2024 (ca. 3%) der Modelle aus dem Datensatz Tritt dieser Fehler auf. Diese Modelle werden ebenfalls nicht klassifiziert.

5.3 Nutzung der automatisierten Klassifikation

Um mit dem Werkzeug BPMN-Modelle zu klassifizieren, wird der Pfad zu einer Datei oder einem Ordner als Parameter übergeben. Die Ergebnisse werden als csv-Datei ausgegeben und zusätzlich wird ein Errorlog geführt. Die Ergebnis-Datei enthält für jedes Diagramm, zusätzlich zu dem Dateinamen und dem Fluss-Layout des analysierten BPMN-Modells, weitere Informationen über den Klassifikationsprozess. So werden zum Beispiel die diskretisierten, vereinfachten Vektorketten und das Fluss-Layout je analysiertem Pfad angegeben. Für Modelle, die nicht analysiert werden können, wird aufgezeigt, aus welchem Grund dies nicht möglich ist. Außerdem wird die benötigte Klassifikationszeit für jedes Modell gespeichert.

Kapitel 6

Validierung

Das Ziel dieser Arbeit ist es, automatisiert das Fluss-Layout von BPMN-Diagrammen auf Grundlage einer nachvollziehbaren Formalisierung zu klassifizieren. Die Automatisierung sollte vorhersehbar sein und genutzt werden können, um große Datensätze zu analysieren. Um die Erreichung dieses Ziels zu überprüfen, werden im vorliegenden Kapitel die Klassifikationen des in Kapitel 5 vorgestellten Werkzeugs und die zugrundeliegenden Formalisierung (siehe Kapitel 4) mit Hilfe zweier Methoden validiert. Zunächst wird hierfür die manuelle Klassifikation von 5297 Diagrammen [25] mit der automatisierten verglichen, bevor die automatisierte Klassifikation von 67169 BPMN-Modellen analysiert wird. Die Methoden haben unterschiedliche Ziele. Das Erkenntnisinteresse der ersten Methode liegt in der Frage, für welche Diagramme und aus welchen Gründen Diskrepanzen zwischen der manuellen und der automatisierten Klassifikation vorliegen. Auf diese Weise kann bewertet werden, ob die vorgestellte Automatisierung eine aufwändige manuelle Betrachtung von BPMN-Diagrammen ersetzen kann. Die zweite Methode soll zeigen, ob die Automatisierung für die Klassifikation großer Datensätze geeignet ist.

6.1 Vergleich zu manueller Klassifikation

Nachfolgend wird Lübke und Wutkes [25] manuelle Klassifikation von 5297 Diagrammen, mit der in dieser Arbeit vorgestellten automatisierten Klassifikation verglichen. Hierbei ist zu beachten, dass die vorgestellte Formalisierung und Automatisierung nicht aud eine präzise Bestätigung von Lübke und Wutkes Arbeit abzielt, sodass divergierende Ergebnisse nicht notwendigerweise als Fehler einer Seite interpretiert werden. Vielmehr ist das Ziel, Gründe für Abweichungen herauszuarbeiten, um den Erwartungshorizont automatisierter Klassifikationen von jenem Manueller abzugrenzen. Hierfür werden zunächst die Klassifikationen anhand von Konfusionsmatrizen verglichen, bevor die Gründe für abweichende Zuordnungen anhand von

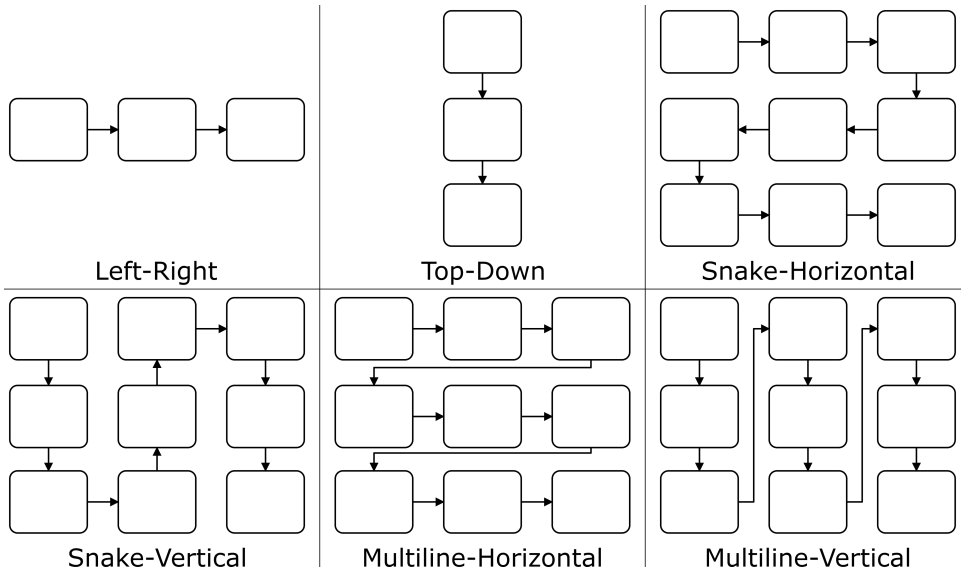

Abbildung 6.1: Von Lübke und Wutke unterschiedene Fluss-Layouts. Abbildung nach Fig. 2 aus Analysis of Prevalent BPMN Layout Choices on GitHub. [25]

Beispieldiagrammen beschrieben werden.

6.1.1 Ergebnisse der Klassifikationen

Um Vergleichbarkeit herzustellen, müssen zunächst die betrachteten Fluss-Layouts abgeglichen werden. Die oben genannten Autoren unterscheiden sechs Fluss-Layouts, die in Abb. 6.1 mit Beispielen vorgestellt werden. Basierend auf dieser Darstellung wird für den Vergleich festgelegt, dass Left-Right dem in dieser Arbeit vorgestellten Fluss-Layout Straight-E in allen Varianten entspricht. Analog ist Straight-S das Fluss-Layout, welches Top-Down am besten repräsentiert. Da von den Autoren nicht genauer spezifiziert wird, welche Anforderungen für ein bestimmtes Fluss-Layout erfüllt sein müssen, werden sowohl Snake-ES als auch Snake-EN in allen Varianten mit dem von Lübke und Wutke betrachteten Layout Snake-Horizontal gleichgesetzt. Diesem Prinzip folgend entspricht Snake-Vertical den Fluss-Layouts Snake-SE und Snake-NE. Außerdem wird Multiline-Horizontal mit Multiline-ES und Multiline-EN verglichen sowie Multiline-Vertical mit Multiline-SE und Multiline-NE. Die Fluss-Layouts L, Stairs, U, Z, Straight-N und Straight-W werden von Lübke und Wutke nicht betrachtet.

Die Abbildungen 6.2, 6.3 und 6.4 zeigen das Ergebnis des Vergleichs als Konfusionsmatrix. In Abb. 6.2 ist durch die nicht normalisierte Darstellungsweise zu erkennen, wie sich die Verteilung der Fluss-Layouts je Klassifikation für diesen Datensatz darstellt. Es wird deutlich, dass der überwältigende

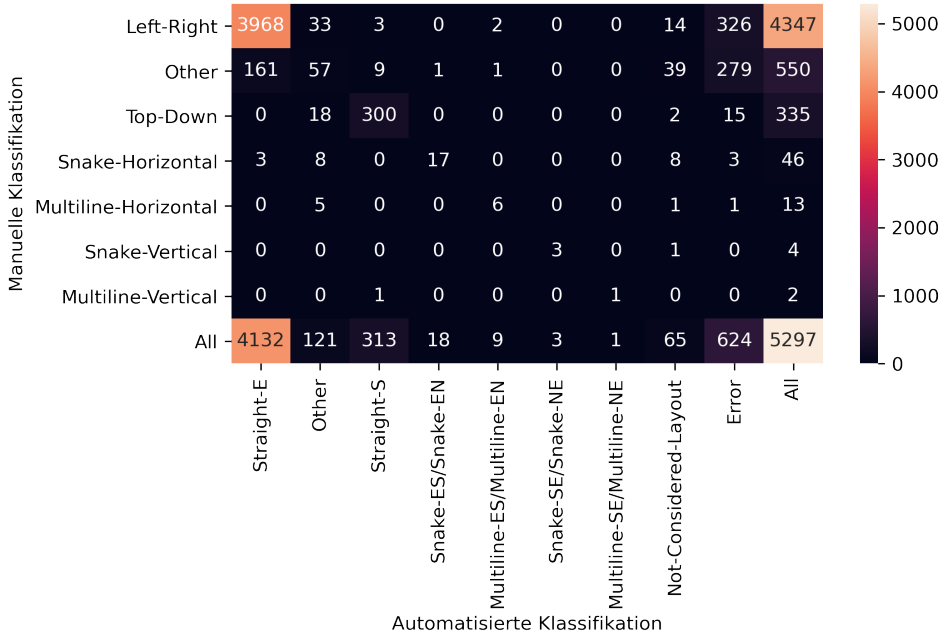

Abbildung 6.2: Manuelle Klassifikation von 5297 Diagrammen im Vergleich zur automatisierten Klassifikation. Nicht normalisierte Konfusionsmatrix.

Teil der Diagramme (manuell $4347/5297 \approx 82\%$, automatisiert $4133/5297 \approx 78\%$) von links nach rechts gerichtet sind, also als Left-Right bzw. Straight-E klassifiziert wurden. Außerdem fällt auf, dass die automatisierte Variante einen großen Anteil der Diagramme ($623/5297 \approx 12\%$) nicht klassifizieren konnte, da ein Error aufgetreten ist (siehe Abschnitt 5.2) und dass ein im Verhältnis zur Diagrammanzahl kleiner ($65/5297 \approx 1\%$) Anteil der Diagramme einem von Lübke und Wutke nicht betrachteten Fluss-Layout (Not-Considered-Layout) zugeordnet wurde. Dies wirft die Frage auf, ob die Not-Considered-Layouts überhaupt oft genug vorkommen, um betrachtungswert zu sein. Allerdings ist bei genauerer Betrachtung zu erkennen, dass in der manuellen Klassifikation nur ein ähnlich kleiner Anteil ($(46 + 13 + 4 + 2)/5297 \approx 1\%$) der Diagramme einem anderen Fluss-Layout als Left-Right oder Top-Down zugeordnet wurde. Demnach ist klar, dass die Verteilung auf die unterschiedlichen Fluss-Layouts dermaßen ungleich verteilt ist, dass, falls über Straight-E oder Straight-S hinausgehende Fluss-Layouts unterschieden werden sollen, eine sehr kleine Anzahl von Diagrammen zu erwarten ist.

Abbildung 6.3 zeigt die Konfusionsmatrix normalisiert gemäß der manuellen Klassifikation. Die Summe der dargestellten Werte einer Zeile beträgt dementsprechend 100%. Es lassen sich somit Aussagen darüber treffen, wie groß der Anteil der manuell als „x“ klassifizierten Diagramme ist, bei dem die Diagramme automatisiert als „y“ klassifiziert wurden. Hieraus

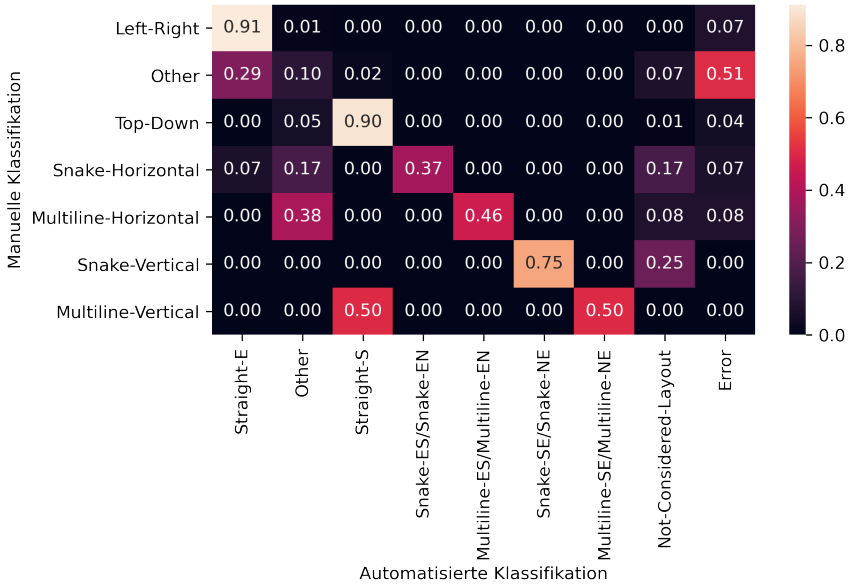

Abbildung 6.3: Manuelle Klassifikation von 5297 Diagrammen im Vergleich zur automatisierten Klassifikation. Nach manueller Klassifikation normalisierte Konfusionsmatrix.

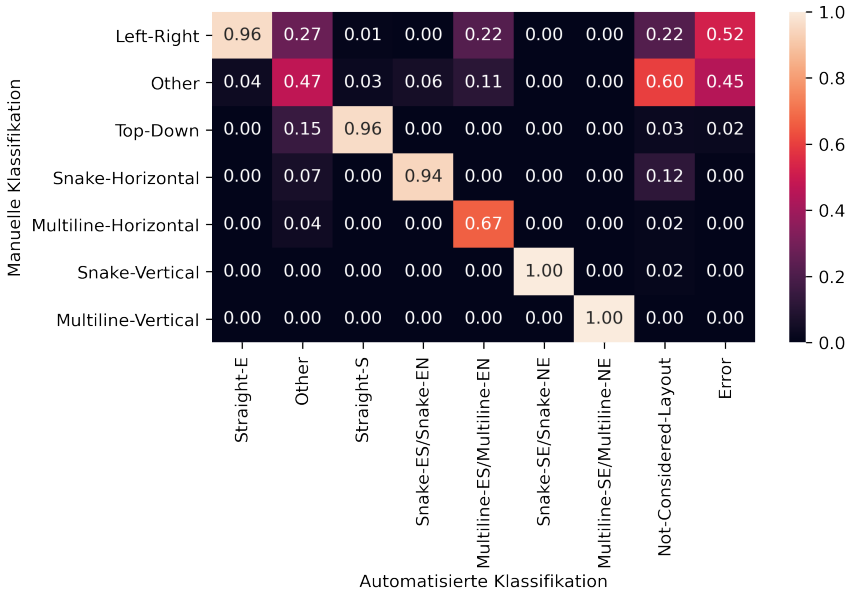

Abbildung 6.4: Manuelle Klassifikation von 5297 Diagrammen im Vergleich zur automatisierten Klassifikation. Nach automatisierter Klassifikation normalisierte Konfusionsmatrix.

lassen sich mehrere Erkenntnisse ableiten. Zum einen ist zu erkennen, dass die Klassifikationen für die geraden Diagramme, also denen mit dem Fluss-Layout Left-Right/Straight-E und Top-Down/Straight-S, sehr ähnlich ausfallen. So wurden 91% der manuell als Left-Right klassifizierten Diagramme automatisiert als Straight-E und analog 90% der manuell als Top-Down klassifizierten als Straight-S klassifiziert. Da 7% bzw. 4% dieser Diagramme jedoch aufgrund von Fehlern nicht automatisiert klassifiziert werden konnten, ist die jeweilige Übereinstimmung sogar höher einzuschätzen. Außerdem wird klar, dass viele (51%) der Diagramme, die manuell keiner der betrachteten Fluss-Layouts zugeordnet werden konnten, bei der automatisierten Klassifikation einen Error hervorrufen. Die automatisierte Klassifikation scheint die Manuelle selten zu bestätigen, wenn sich manuell für Snake-Horizontal, Multiline-Horizontal oder Snake-Vertical entschieden wurde.

Schließlich zeigt Abb. 6.4 die Konfusionsmatrix normalisiert nach der automatisierten Klassifikation. Es ist zu erkennen, dass sie in den meisten Fällen mit der manuellen Klassifikation übereinstimmt, wenn automatisiert eine der betrachteten Fluss-Layouts identifiziert werden konnte. Der niedrigste Übereinstimmungswert in diesem Sinne liegt bei den horizontalen Multiline Varianten vor und ist mit 67% deutlich über 50%. Allerdings ist festzustellen, dass viele (52%) der automatisiert als analysierbar (kein Error) aber nicht klassifizierbar (Other) eingestuften Diagramme manuell einem Fluss-Layout zugeordnet werden konnten.

6.1.2 Gründe für Abweichungen der Klassifikationen

Nachdem dargestellt wurde, wie sich die manuelle von der automatisierten Klassifikation unterscheidet, werden an dieser Stelle Gründe untersucht. Bei der Betrachtung aller unterschiedlich klassifizierten Diagramme konnten mehrere Gründe identifiziert werden, die Einfluss auf die Interpretation von späteren Klassifikationen haben können. Manuelle Klassifikation erlaubt es intuitiv Layouts zu vervollständigen, z. B. in Abb. 6.5 zu sehen ist. Das dargestellte Diagramm wurde manuell als Snake-Horizontal klassifiziert. Dies ist möglich, indem intuitiv die fehlenden Sequenz-Flüsse ergänzt wurden. Die automatisierte Klassifikation hingegen versucht nicht, das Diagramm zu vervollständigen, sodass das Diagramm automatisiert als Straight-E-Clean klassifiziert wird.

Ein anderer Grund für Diskrepanzen zwischen den Klassifikationen ist die Fehleranfälligkeit manueller Klassifikation. So wurde das in Abb. 6.6 dargestellte Diagramm manuell als Left-Right klassifiziert, obwohl das Fluss-Layout von Oben nach Unten gerichtet ist. Die automatisierte Klassifikation (Straight-S-Clean) erkennt die Ausrichtung korrekt.

Aufgrund der unklaren Anforderungen an bestimmte Fluss-Layouts wirkt die manuelle Klassifikation in manchen Fällen inkonsistent. Bspw. sind

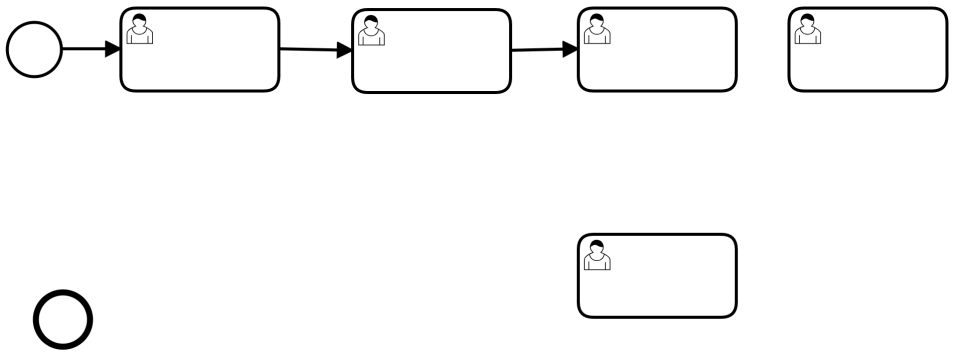

Abbildung 6.5: Manuell als Snake-Horizontal, automatisch als Straight-E-Clean klassifiziertes Diagramm. Automatisierung vervollständigt Diagramme nicht.

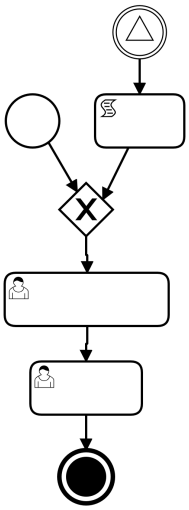

Abbildung 6.6: Manuell als Left-Right, automatisch als Straight-S-Clean klassifiziertes Diagramm. Menschliche Fehler.

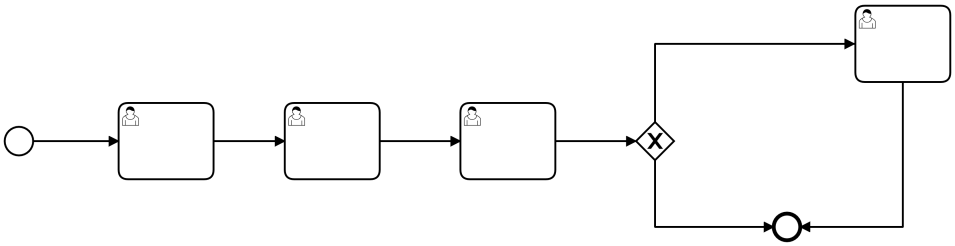

Abbildung 6.7: Manuell als Other, automatisch als Straight-E klassifiziertes Diagramm. Manuelle Klassifikation teilweise sehr streng.

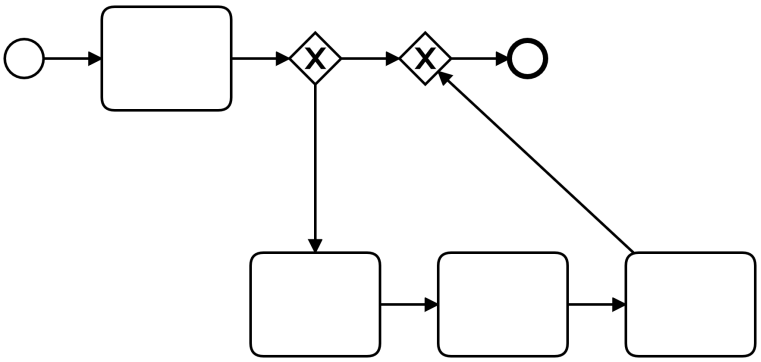

Abbildung 6.8: Manuell als Left-Right, automatisch als Other klassifiziertes Diagramm. Manuelle Klassifikation erlaubt teilweise viel Spielraum

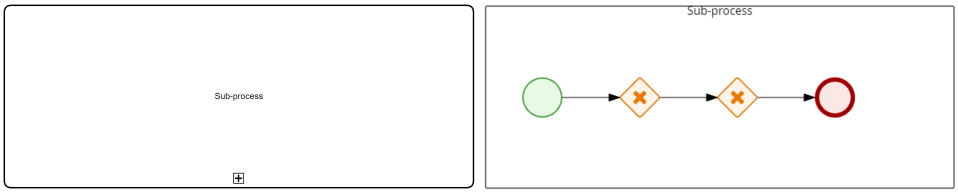

Abbildung 6.9: Manuell als Other, automatisch als Straight-E-Clean klassifiziertes Diagramm. Links png-Export mit BPMN.io, rechts png-Export mit jBPM. Diagramm-Editor zum Sichten der Diagramme hat Einfluss auf Klassifikation.

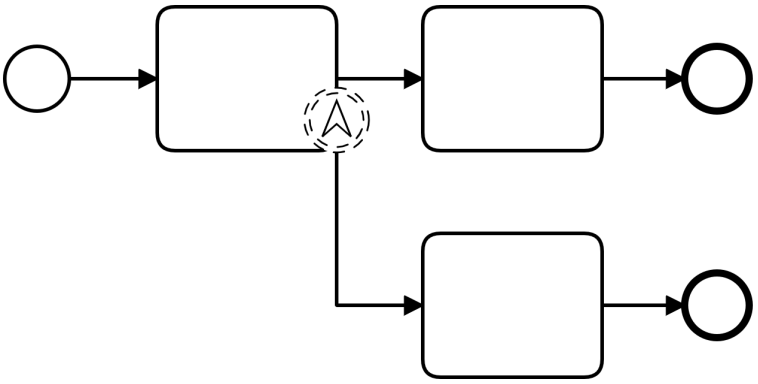

Abbildung 6.10: Manuell als Left-Right, automatisch als Other klassifiziertes Diagramm. Möglicherweise zu viele Annahmen vorausgesetzt für Formalisierung.

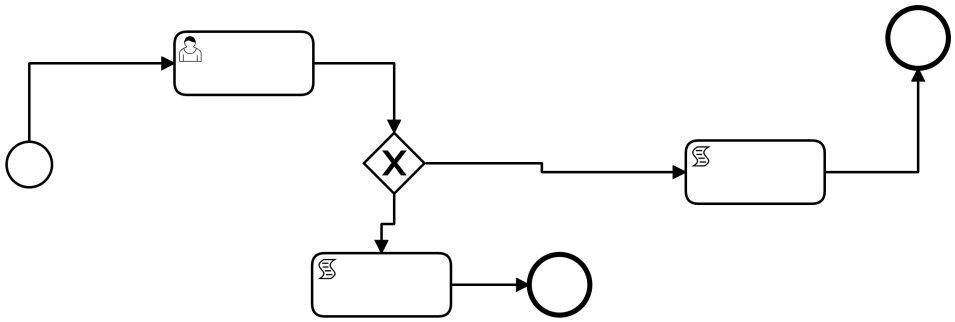

Abbildung 6.11: Manuell als Left-Right, automatisch als Other klassifiziertes Diagramm. Formalisierung lässt wenig Spielraum.

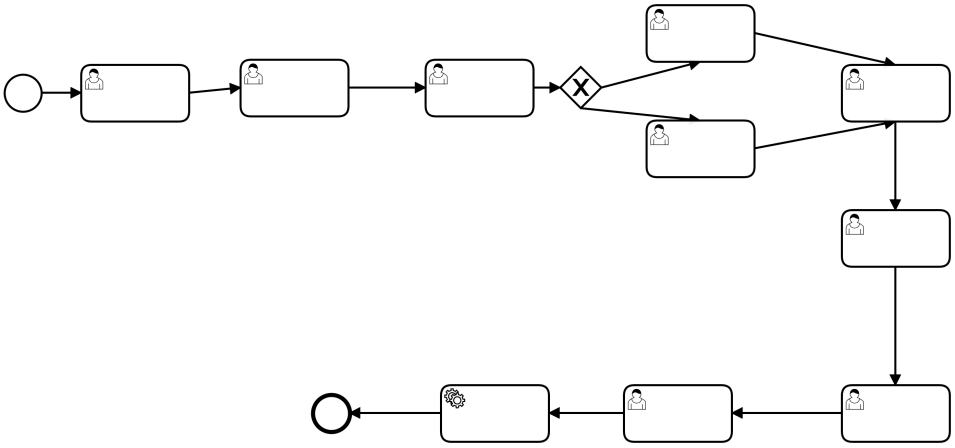

Abbildung 6.12: Manuell als Snake-Horizontal, automatisch als U-ES klassifiziertes Diagramm. Unterschiedliche Menge der betrachteten Fluss-Layouts.

Lübke und Wutke bei manchen Diagrammen sehr streng, wie in Abb. 6.7 zu sehen. Während das Diagramm manuell als nicht klassifizierbar (Other) eingestuft wurde, betrachtet die automatisierte Klassifikation die zwei Pfade einzeln und klassifiziert sie als Straight-E-Clean und Straight-E-LEndRight, sodass sie zu dem Diagramm-Layout Straight-E zusammengefügt werden. Andererseits lässt die manuelle Klassifikation teilweise mehr Spielraum. So wurde das Diagramm aus Abb. 6.8 manuell als Left-Right klassifiziert. Es ist möglich, dass die Autoren eine gute Begründung für ihre Klassifikation haben, doch ist die Nachvollziehbarkeit durch die fehlende Formalisierung stark eingeschränkt.

Abbildung 6.9 beweist den Einfluss des zur Betrachtung der Diagramme genutzten Editors auf die manuelle Klassifikation. So zeigt BPMN.iO den Inhalt des Sub-Prozesses nicht, jBPMN hingegen schon. Lübke und Wutke basieren ihre (Other) Klassifikation auf der links dargestellten alleinstehenden Aktivität, während die automatisierte Klassifikation alle Pfade des Models (auch die in Sub-Prozessen) betrachtet und somit Straight-E-Clean ausgibt.

Allerdings sind auch Annahmen, die für die Formalisierung in dieser Arbeit getroffen wurden, ein Grund für abweichende Klassifikationen. Das Diagramm in Abb. 6.10 wird automatisiert als nicht klassifizierbar eingestuft, da der Pfad mit dem Boundary-Ereignis als Z-ES-Clean klassifiziert wird, sodass zwei Pfade ein unterschiedliches Fluss-Layout haben und kein Layout in mehr als 2/3 aller Pfade vorkommt. Der Grund für diese Klassifikation als Z-ES-Clean findet sich in dem Boundary-Ereignis, welches nicht wie erwartet unten, sondern rechts an der Aktivität angehängt ist. Folglich wird die vertikale Komponente des aus dem Boundary-Ereignis ausgehendem Sequenz-Fluss nicht entfernt.

Abschließend ist die automatisierte Klassifikation bei der Analyse von Diagrammen mit unsauberen Layouts vergleichsweise streng. Abbildung 6.11 zeigt ein Diagramm, das zwar nicht gerade nach rechts verläuft, aber trotzdem manuell als Left-Right klassifizierbar ist. Automatisiert war aufgrund der vielen Richtungswechsel kein Fluss-Layout feststellbar. Wie oben bereits angemerkt, betrachten Lübke und Wutke nicht die gleichen Fluss-Layouts wie diese Arbeit. Deshalb entstehen, wie z. B. in Abb. 6.12 zu sehen, Abweichungen zwischen den Klassifikationen. Das gezeigte Diagramm ist entsprechend der in dieser Arbeit vorgestellten Formalisierung ein U-ES-Layout, da sich auf dem vertikalen Verbindungsstück zwischen den beiden Zeilen Fluss-Objekte befinden. Manuell wurden keine U-Layouts betrachtet, sodass das Diagramm als Snake-Horizontal klassifiziert wurde.

6.2 Anwendung auf großen Datensatz

Um zu überprüfen, ob das Klassifikationswerkzeug geeignet ist große Datensätze zu analysieren, wird der große GitHub Datensatz (siehe Abschnitt 4.2) mit 48679 klassifizierbaren Diagrammen genutzt.

Auf einem Desktop PC mit AMD Ryzen 5 3600 CPU benötigt die Klassifikation aller Modelle ca. zwei Stunden. Allerdings ist zu beachten, dass große Unterschiede im Zeitaufwand pro Modell bestehen. So wurde auf die Klassifikation der langsamsten 10 Modelle benötigt ca. 99% der Zeit aufgewand, während das Diagramm mit der elft-längsten Klassifikationszeit innerhalb von weniger als 30 Sekunden klassifiziert wurde.

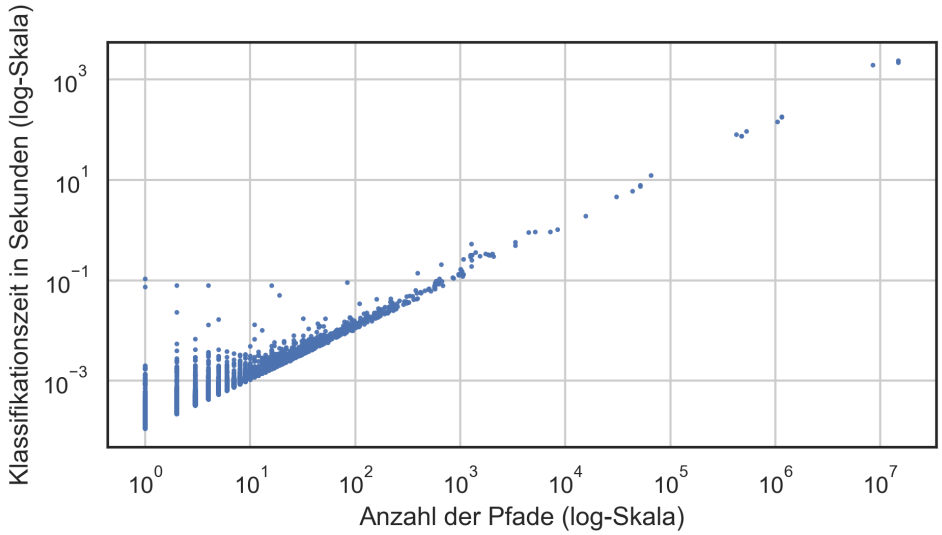

Abbildung 6.13: Klassifikationszeit je Modell aus GitHub Datensatz. Doppelt-logarithmische Darstellung des Zusammenhangs zwischen der Anzahl der Pfade im Modell und der Klassifikationszeit.

Abbildung 6.13 zeigt, dass die Laufzeit der Klassifikation maßgeblich von der Anzahl der Pfade in dem analysierten Modell abhängt. Es scheint ein Potenzgesetz vorzuliegen, da die Daten in der doppelt-logarithmischen Darstellung eine Gerade bilden.

Das Histogramm, das in Abb. 6.14 dargestellt ist, illustriert noch deutlicher als Abb. 6.13, die allermeisten Diagramme des Datensatzes wenige Pfade aufweisen. Dies erklärt den großen Anteil an der Klassifikationszeit, den sehr wenige Diagramme einnehmen. Beachtenswert ist, dass die maximal mögliche Anzahl schleifenfreier Pfade durch die Fakultät der Anzahl von Fluss-Objekten abgeschätzt werden kann. Das Diagramm mit der höchsten Anzahl von Pfaden in dem Datensatz weist ca. 15 Millionen Pfade auf. Dieser Wert ist theoretisch mit einem Modell mit nur 14 Fluss-Objekten

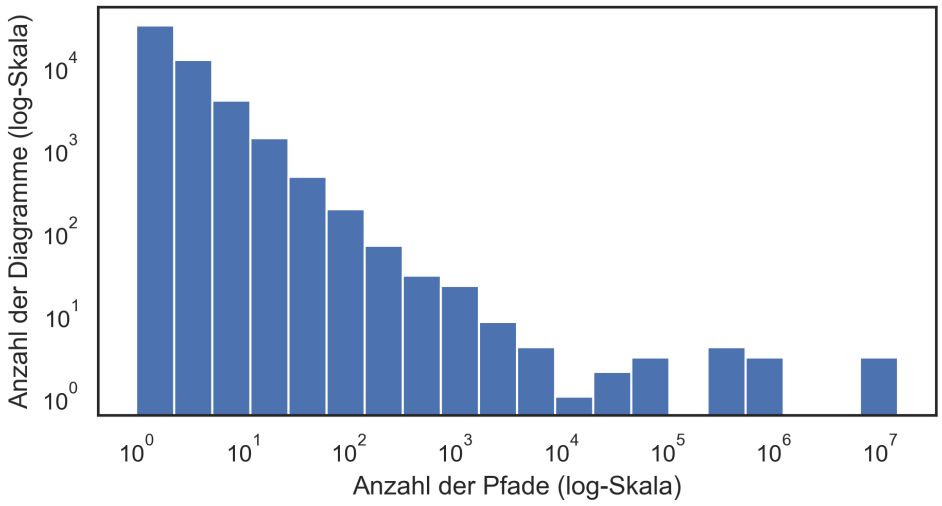

Abbildung 6.14: Anzahl der Diagramme mit bestimmter Anzahl von Pfaden. Doppelt-logarithmisch dargestelltes Histogramm.

deutlich zu übertreffen ($(14-2)! = 47,9 \cdot 10^7$).Hierfür müssten alle 14 Objekte in beide Richtungen durch einen Sequenz-Fluss verbunden sein, bis auf ein Start-Objekt, was keine eingehenden- und ein End-Objekt, was keine ausgehenden Sequenz-Flüsse hat. Es lässt sich nicht leicht bestimmen, wie lange die Analyse eines Modells dauert, sodass zu einer Etablierung eines Zeitlimits geraten wird, um die Gesamt-Klassifikations-Zeit zu reduzieren.

Um die Nützlichkeit des Werkzeugs besser einschätzen zu können, wird überprüft, welche Aussagen über den Datensatz basierend auf der automatisierten Klassifikation getroffen werden können. Abbildung 6.15 zeigt die Verteilung der Grund-Flus-Layouts. Die mit Abstand meisten Diagramme haben weisen ein Straight-Layout (man beachte die logarithmische Skala) auf. Abbildung 6.16 zeigt die Verteilung der Fluss-Layouts für jene Diagramme, für die ein Straight-Layout festgestellt wurde. Es ist zu erkennen, dass der Orientierung E in der Variante Clean am meisten Diagramme zugeordnet wurden. Präzisiert wurden $\frac{34274}{43988} \approx 78\%$ aller Diagramme mit Straight Layouts und somit $\frac{34274}{48675} \approx 70\%$ aller analysierbaren Diagramme mit Straight-E-Clean klassifiziert. Der Vollständigkeit wegen sind die Ergebnisse für die anderen Grund-Layouts in Anhang B in Abb. B.1, B.2, B.3, B.4, B.5 und B.6 zu finden.

6.3 Ergebnisse/ Diskussion

Die Validierung beweist, dass die formalisierte automatisierte Klassifikation genutzt werden kann, um große Datensätze zu analysieren, wobei

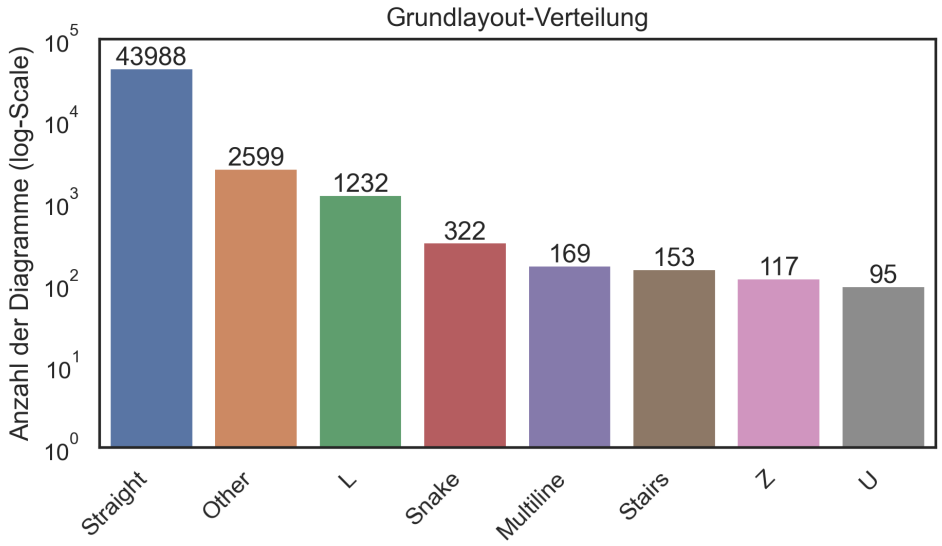

Abbildung 6.15: Verteilung der Flow-Layouts

mit manueller Klassifikation vergleichbare Ergebnisse erzielt werden. Die Automatisierung auf Basis der Formalisierung bietet nebst geringerem Arbeitsaufwand einige Vorteile gegenüber der manuellen Klassifikation. So werden Flüchtigkeitsfehler und der subjektiven Wahrnehmung geschuldete Inkonsistenzen vermieden. Allerdings bietet auch die manuelle Klassifikation derzeit Vorteile gegenüber der Automatisierten. Z. B. ist das Werkzeug, anders als Nutzer mit BPMN-Erfahrung, nicht in der Lage, unvollständige Diagramme intuitiv zu vervollständigen um eine genauere Klassifikation des vom Autor angestrebten Layouts zu ermöglichen. Außerdem werden automatisiert bestimmte Diagramme nicht der Erwartung entsprechend klassifiziert, da sie Annahmen, die in der Formalisierung getroffen wurden, nicht erfüllen. Zudem wurde festgestellt, dass unsaubere Diagramme manuell besser klassifiziert werden können.

Zusammengefasst ist das Werkzeug geeignet, um große Datensätze halbautomatisiert zu klassifizieren. Modelle, die sehr viele Pfade beinhalten, können nicht schnell klassifiziert werden, sodass in diesem Fall eine manuelle Inspektion besser geeignet ist. In dem betrachteten Datensatz befinden sich allerdings kaum Diagramme, deren Klassifikation einen großen Zeitaufwand benötigt. Wenn bspw. ein Zeitlimit von 30 Sekunden pro Modell gesetzt worden wäre, hätten nur 11 der 48679 Diagramme nicht klassifiziert werden können. Die Gesamtzeit wäre in diesem Fall von ca. zwei Stunden auf ca. fünf Minuten reduziert worden. Die Ergebnisse der Klassifikation können als CSV-Datei einfach ausgewertet werden.

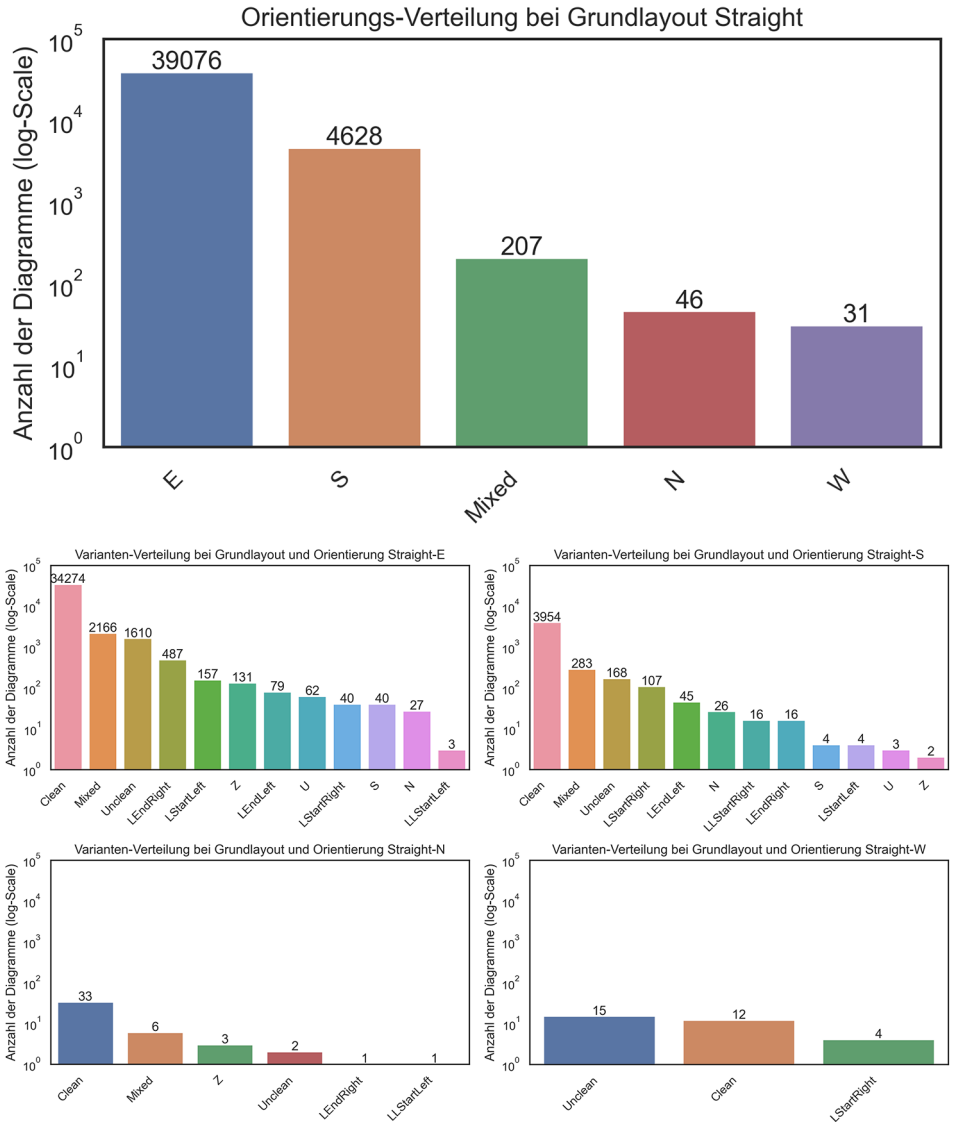

Abbildung 6.16: Verteilung der Flow-Layouts bei Grundlayout Straight

Kapitel 7

Fazit und Ausblick

In dieser Masterarbeit wurde jener Aspekt des Layouts von BPMN-Diagrammen behandelt, der die Struktur der Elemente beschreibt. Indem eine Fluss-Layout Hierarchie ausgearbeitet und die einzelnen Fluss-Layouts formal definiert wurden, konnte ein Algorithmus zur Klassifikation von Fluss-Layouts konstruiert und in einem einem Werkzeug implementiert werden.

Die Validierung hat gezeigt, dass die Automatisierung genutzt werden kann große Datensätze zu analysieren und dabei ähnliche Ergebnisse erzielt wie die mit großem Arbeitsaufwand verbundene manuelle Klassifikation. Da das vorgestellte Werkzeug Diagramme auf Basis der Formalisierung regelbasiert klassifiziert, können die Ergebnisse Objektivität Nachvollziehbarkeit und Reproduzierbarkeit beanspruchen. Im Kontext der BPMN-Verständlichkeits-Forschung bedeutet dies einen Meilenstein, da bis dato keinerlei verlässliche Standardisierung eingeführt wurde.

Obwohl die vorgestellten Konzepte funktionieren und ein zufriedenstellendes Ergebnis erzielen, sollten sie weiterentwickelt werden, um möglichst viele Diagramm-Layouts genau beschreiben zu können. Die Grundidee, nur schleifenfreie Pfade von Start- zu End-Objekten zu analysieren, schränkt die Klassifikationsmöglichkeiten ein. Dies ist z. B. bei solchen Diagrammen der Fall, die kein Start- oder End-Objekt aufweisen oder die Struktur der Elemente, die in Schleifen vorkommen relevant ist. Auch die einzelnen Module des Algorithmus (und die entsprechenden Funktionen der Formalisierung) bieten Verbesserungsmöglichkeiten. So könnte die Umwandlung der Layoutpfade in Vektorketten allgemeingültiger vorgenommen werden, indem die Regeln unabhängig von Annahmen wie denen zur Positionierung von Boundary-Ereignissen formuliert werden. Auch die Diskretisierung der Vektor-Richtungen sollte in Zukunft genau analysiert werden um Optimierungspotenzial auszuschöpfen indem z. B. mehr oder weniger diskrete Richtungen, oder gar unterschiedlich große Richtungbereiche je diskreter Richtung, betrachtet werden. Letztendlich sind die regulären Ausdrücke die

Stellschraube mit der die Klassifikation am genausten kontrolliert werden kann.

Zukünftige Forschungsvorhaben könnten an diese Arbeit anknüpfen, indem hier vorgestellte Konzepte auf andere Graph-Basierte-Diagrammtypen, wie z. B. UML-Diagramme, übertragen werden. Die in dieser Abhandlung vorgestellten Fluss-Layouts sind zwar BPMN-Spezifisch, doch könnte z. B. die Idee der hierarchischen Einordnung von häufig genutzten Strukturen auch in anderen Umgebungen hilfreich sein.

Zusammengefasst lässt sich sagen, dass das vorgestellte Konzept, Fluss Layouts mit Hilfe von regulären Ausdrücken und diskretisierten, vereinfachten Vektorketten zu formalisieren und klassifizieren, vielversprechend ist. Die Modulare Struktur ist leicht verständlich und ermöglicht die Weiterentwicklung einzelner Komponenten. Die Formalisierung und die automatisierte Klassifikation sind wichtige Beiträge zur BPMN-Verständlichkeits-Forschung und bilden die Grundlage für weitere Werkzeuge, die es BPMN-Nutzern in der Zukunft erleichtern können Verständliche Diagramme zu erstellen.

Anhang A

Reguläre Ausdrücke

Abkürzung	Bedeutung
E-M	E-M-L|E-M-S
E-NM	E-NM-L|E-NM-S
E-L	E-M-L|E-NM-L
E-S	E-M-S|E-NM-S
E	E-M|E-NM
S-M	S-M-L|S-M-S
S-NM	S-NM-L|S-NM-S
S-L	S-M-L|S-NM-L
S-S	S-M-S|S-NM-S
S	S-M|S-NM
W-M	W-M-L|W-M-S
W-NM	W-NM-L|W-NM-S
W-L	W-M-L|W-NM-L
W-S	W-M-S|W-NM-S
W	W-M|W-NM
N-M	N-M-L|N-M-S
N-NM	N-NM-L|N-NM-S
N-L	N-M-L|N-NM-L
N-S	N-M-S|N-NM-S
N	N-M|N-NM
NE-M	NE-M-L|NE-M-S
NE-NM	NE-NM-L|NE-NM-S
NE-L	NE-M-L|NE-NM-L
NE-S	NE-M-S|NE-NM-S
NE	NE-M|NE-NM
SE-M	SE-M-L|SE-M-S
SE-NM	SE-NM-L|SE-NM-S
SE-L	SE-M-L|SE-NM-L
SE-S	SE-M-S|SE-NM-S
SE	SE-M|SE-NM
SW-M	SW-M-L|SW-M-S
SW-NM	SW-NM-L|SW-NM-S
SW-L	SW-M-L|SW-NM-L
SW-S	SW-M-S|SW-NM-S
SW	SW-M|SW-NM

NW-M	NW-M-L|NW-M-S
NW-NM	NW-NM-L|NW-NM-S
NW-L	NW-M-L|NW-NM-L
NW-S	NW-M-S|NW-NM-S
NW	NW-M|NW-NM
NNE-M	NNE-M-L|NNE-M-S
NNE-NM	NNE-NM-L|NNE-NM-S
NNE-L	NNE-M-L|NNE-NM-L
NNE-S	NNE-M-S|NNE-NM-S
NNE	NNE-M|NNE-NM
NEE-M	NEE-M-L|NEE-M-S
NEE-NM	NEE-NM-L|NEE-NM-S
NEE-L	NEE-M-L|NEE-NM-L
NEE-S	NEE-M-S|NEE-NM-S
NEE	NEE-M|NEE-NM
SEE-M	SEE-M-L|SEE-M-S
SEE-NM	SEE-NM-L|SEE-NM-S
SEE-L	SEE-M-L|SEE-NM-L
SEE-S	SEE-M-S|SEE-NM-S
SEE	SEE-M|SEE-NM
SSE-M	SSE-M-L|SSE-M-S
SSE-NM	SSE-NM-L|SSE-NM-S
SSE-L	SSE-M-L|SSE-NM-L
SSE-S	SSE-M-S|SSE-NM-S
SSE	SSE-M|SSE-NM
SSW-M	SSW-M-L|SSW-M-S
SSW-NM	SSW-NM-L|SSW-NM-S
SSW-L	SSW-M-L|SSW-NM-L
SSW-S	SSW-M-S|SSW-NM-S
SSW	SSW-M|SSW-NM
SWW-M	SWW-M-L|SWW-M-S
SWW-NM	SWW-NM-L|SWW-NM-S
SWW-L	SWW-M-L|SWW-NM-L
SWW-S	SWW-M-S|SWW-NM-S
SWW	SWW-M|SWW-NM
NWW-M	NWW-M-L|NWW-M-S
NWW-NM	NWW-NM-L|NWW-NM-S
NWW-L	NWW-M-L|NWW-NM-L
NWW-S	NWW-M-S|NWW-NM-S
NWW	NWW-M|NWW-NM
NNW-M	NNW-M-L|NNW-M-S
NNW-NM	NNW-NM-L|NNW-NM-S
NNW-L	NNW-M-L|NNW-NM-L
NNW-S	NNW-M-S|NNW-NM-S
NNW	NNW-M|NNW-NM
SEE-TO-S-NM	SEE-NM|SE-NM|SSE-NM|S-NM
NEE-TO-N-NM	NEE-NM|NE-NM|NNE-NM|N-NM
SSW-TO-W-NM	SSW-NM|SW-NM|SWW-NM|W-NM
SSE-TO-E-NM	SSE-NM|SE-NM|SEE-NM|E-NM
NWW-TO-N-NM	NWW-NM|NW-NM|NNW-NM|N-NM
SWW-TO-S-NM	SWW-NM|SW-NM|SSW-NM|S-NM
NNE-TO-E-NM	NNE-NM|NE-NM|NEE-NM|E-NM
NNW-TO-W-NM	NNW-NM|NW-NM|NWW-NM|W-NM

SEE-TO-S-NM-L	SEE-NM-L|SE-NM-L|SSE-NM-L|S-NM-L
NEE-TO-N-NM-L	NEE-NM-L|NE-NM-L|NNE-NM-L|N-NM-L
SSW-TO-W-NM-L	SSW-NM-L|SW-NM-L|SWW-NM-L|W-NM-L
SSE-TO-E-NM-L	SSE-NM-L|SE-NM-L|SEE-NM-L|E-NM-L
NWW-TO-N-NM-L	NWW-NM-L|NW-NM-L|NNW-NM-L|N-NM-L
SWW-TO-S-NM-L	SWW-NM-L|SW-NM-L|SSW-NM-L|S-NM-L
NNE-TO-E-NM-L	NNE-NM-L|NE-NM-L|NEE-NM-L|E-NM-L
NNW-TO-W-NM-L	NNW-NM-L|NW-NM-L|NWW-NM-L|W-NM-L
SEE-TO-SSE-NM	SEE-NM|SE-NM|SSE-NM
NEE-TO-NNE-NM	NEE-NM|NE-NM|NNE-NM
SSW-TO-SWW-NM	SSW-NM|SW-NM|SWW-NM
SSE-TO-SEE-NM	SSE-NM|SE-NM|SEE-NM
NWW-TO-NNW-NM	NWW-NM|NW-NM|NNW-NM
SWW-TO-SSW-NM	SWW-NM|SW-NM|SSW-NM
NNE-TO-NEE-NM	NNE-NM|NE-NM|NEE-NM
NNW-TO-NWW-NM	NNW-NM|NW-NM|NWW-NM
SEE-TO-S-M	SEE-M|SE-M|SSE-M|S-M
NEE-TO-N-M	NEE-M|NE-M|NNE-M|N-M
SSW-TO-W-M	SSW-M|SW-M|SWW-M|W-M
SSE-TO-E-M	SSE-M|SE-M|SEE-M|E-M
NWW-TO-N-M	NWW-M|NW-M|NNW-M|N-M
SWW-TO-S-M	SWW-M|SW-M|SSW-M|S-M
NNE-TO-E-M	NNE-M|NE-M|NEE-M|E-M
NNW-TO-W-M	NNW-M|NW-M|NWW-M|W-M
SEE-TO-S	SEE-TO-S-M|SEE-TO-S-NM
NEE-TO-N	NEE-TO-N-M|NEE-TO-N-NM
SSW-TO-W	SSW-TO-W-M|SSW-TO-W-NM
SSE-TO-E	SSE-TO-E-M|SSE-TO-E-NM
NWW-TO-N	NWW-TO-N-M|NWW-TO-N-NM
SWW-TO-S	SWW-TO-S-M|SWW-TO-S-NM
NNE-TO-E	NNE-TO-E-M|NNE-TO-E-NM
NNW-TO-W	NNW-TO-W-M|NNW-TO-W-NM
S-45-NM	S-NM|SSE-NM|SSW-NM
N-45-NM	N-NM|NNE-NM|NNW-NM
E-45-NM	E-NM|NEE-NM|SEE-NM
W-45-NM	W-NM|NWW-NM|SWW-NM
W-45-NM-S	W-NM-S|SWW-NM-S|NWW-NM-S
S-45-NM-S	S-NM-S|SSE-NM-S|SSW-NM-S
N-45-NM-S	N-NM-S|NNE-NM-S|NNW-NM-S
E-45-NM-S	E-NM-S|NEE-NM-S|SEE-NM-S
W-45-NM-L	W-NM-L|SWW-NM-L|NWW-NM-L
S-45-NM-L	S-NM-L|SSE-NM-L|SSW-NM-L
N-45-NM-L	N-NM-L|NNE-NM-L|NNW-NM-L
E-45-NM-L	E-NM-L|NEE-NM-L|SEE-NM-L
S-45-M	S-M|SSE-M|SSW-M
N-45-M	N-M|NNE-M|NNW-M
E-45-M	E-M|NEE-M|SEE-M
W-45-M	W-M|NWW-M|SWW-M
W-45-M-S	W-M-S|SWW-M-S|NWW-M-S
S-45-M-S	S-M-S|SSE-M-S|SSW-M-S
N-45-M-S	N-M-S|NNE-M-S|NNW-M-S
E-45-M-S	E-M-S|NEE-M-S|SEE-M-S

W-45-M-L	W-M-L|SWW-M-L|NWW-M-L
S-45-M-L	S-M-L|SSE-M-L|SSW-M-L
N-45-M-L	N-M-L|NNE-M-L|NNW-M-L
E-45-M-L	E-M-L|NEE-M-L|SEE-M-L
S-90-NM	S-45-NM|SE-NM|SW-NM
E-90-NM	E-45-NM|NE-NM|SE-NM
N-90-NM	N-45-NM|NE-NM|NW-NM
W-90-NM	W-45-NM|NW-NM|SW-NM
SE-45	SE|SEE|SSE
SW-45	SW|SWW|SSW
NW-45	NW|NWW|NNW
NE-45	NE|NEE|NNE
SM	[S-M]
WM	[W-M]
NM	[N-M]
EM	[E-M]
SNM	[S-NM]
WNM	[W-NM]
NNM	[N-NM]
ENM	[E-NM]

Die restlichen Abkürzungen sind aus Platzgründen auf der nächsten Seite im Querformat dargestellt.

Abkürzung	Bedeutung
ALL-NM-S	N-45-NM-S|NE-NM-S|E-45-NM-S|SE-NM-S|S-45-NM-S|SW-NM-S|W-45-NM-S|NW-NM-S
ALL-NM-L	N-45-NM-L|NE-NM-L|E-45-NM-L|SE-NM-L|S-45-NM-L|SW-NM-L|W-45-NM-L|NW-NM-L
ALL-NM	ALL-NM-S|ALL-NM-L
ALL-M-S	N-45-M-S|NE-M-S|E-45-M-S|SE-M-S|S-45-M-S|SW-M-S|W-45-M-S|NW-M-S
ALL-M-L	N-45-M-L|NE-M-L|E-45-M-L|SE-M-L|S-45-M-L|SW-M-L|W-45-M-L|NW-M-L
ALL-M	ALL-M-S|ALL-M-L
ALL	ALL-M|ALL-NM
ALLOWED-FOR-E	ALL-NM-S|N-NM-L|NNE-NM-L|NE-NM-L|NEE-NM-L|E-NM-L|SEE-NM-L|SE-NM-L|SSE-NM-L|S-NM-L|NEE-M|E-M|SEE-M
ALLOWED-FOR-S	ALL-NM-S|E-NM-L|SEE-NM-L|SE-NM-L|SSE-NM-L|S-NM-L|SSW-NM-L|SW-NM-L|SWW-NM-L|W-NM-L|SSE-M|S-M|SSW-M
ALLOWED-FOR-W	ALL-NM-S|N-NM-L|S-NM-L|SSW-NM-L|SW-NM-L|SWW-NM-L|W-NM-L|NWW-NM-L|NW-NM-L|NNW-NM-L|SWW-M|W-M|NWW-M
ALLOWED-FOR-N	ALL-NM-S|N-NM-L|NNE-NM-L|NE-NM-L|NEE-NM-L|E-NM-L|W-NM-L|NWW-NM-L|NW-NM-L|NNW-NM-L|NNW-M|N-M|NNE-M

Fluss-Layout	Regulärer Ausdruck
STRAIGHT-E-CLEAN	E
STRAIGHT-E-LENDRIGHT	E-M || SEE-TO-S-NM
STRAIGHT-E-LSTARTLEFT	SEE-TO-S-NM || E-M
STRAIGHT-E-LENDLEFT	E-M || NEE-TO-N-NM
STRAIGHT-E-LSTARTRIGHT	NEE-TO-N-NM || E-M
STRAIGHT-E-LLSTARTLEFT	S-45-NM || E-M || S-45-NM
STRAIGHT-E-LLSTARTRIGHT	N-45-NM || E-M || N-45-NM
STRAIGHT-E-U	(SEE-TO-S-NM || E-M || NEE-TO-N-NM) | (SEE-TO-SSE-NM || NEE-TO-NNE-NM)
STRAIGHT-E-N	(NEE-TO-N-NM || E-M || SEE-TO-S-NM) | (NEE-TO-NNE-NM || SEE-TO-SSE-NM)
STRAIGHT-E-Z	E-M || SEE-TO-S-NM || E-M
STRAIGHT-E-S	E-M || NEE-TO-N-NM || E-M
STRAIGHT-E-UNCLEAN	(ALLOWED-FOR-E || *E-M-L || ALLOWED-FOR-E*)+
STRAIGHT-S-CLEAN	S
STRAIGHT-S-LENDRIGHT	S-M || SSW-TO-W-NM
STRAIGHT-S-LSTARTLEFT	SSW-TO-W-NM || S-M
STRAIGHT-S-LENDLEFT	S-M || SSE-TO-E-NM
STRAIGHT-S-LSTARTRIGHT	SSE-TO-E-NM || S-M
STRAIGHT-S-LLSTARTLEFT	W-45-NM || S-M || W-45-NM
STRAIGHT-S-LLSTARTRIGHT	E-45-NM || S-M || E-45-NM
STRAIGHT-S-U	(SSW-TO-W-NM || S-M || SSE-TO-E-NM) | (SSW-TO-SWW-NM || SSE-TO-SEE-NM)
STRAIGHT-S-N	(SSE-TO-E-NM || S-M || SSW-TO-W-NM) | (SSE-TO-SEE-NM || SSW-TO-SWW-NM)
STRAIGHT-S-Z	S-M || SSW-TO-W-NM || S-M
STRAIGHT-S-S	S-M || SSE-TO-E-NM || S-M
STRAIGHT-S-UNCLEAN	(ALLOWED-FOR-S || *S-M-L || ALLOWED-FOR-S*)+
STRAIGHT-W-CLEAN	W
STRAIGHT-W-LENDRIGHT	W-M || NWW-TO-N-NM
STRAIGHT-W-LSTARTLEFT	NWW-TO-N-NM || W-M
STRAIGHT-W-LENDLEFT	W-M || SWW-TO-S-NM
STRAIGHT-W-LSTARTRIGHT	SWW-TO-S-NM || W-M
STRAIGHT-W-LLSTARTLEFT	N-45-NM || W-M || N-45-NM
STRAIGHT-W-LLSTARTRIGHT	S-45-NM || W-M || S-45-NM
STRAIGHT-W-U	(NWW-TO-N-NM || W-M || SWW-TO-S-NM) | (NWW-TO-NNW-NM || SWW-TO-SSW-NM)
STRAIGHT-W-N	(SWW-TO-S-NM || W-M || NWW-TO-N-NM) | (SWW-TO-SSW-NM || NWW-TO-NNW-NM)
STRAIGHT-W-Z	W-M || NWW-TO-N-NM || W-M

STRAIGHT-W-S	[W-M][SWW-TO-S-NM][W-M]
STRAIGHT-W-UNCLEAN	([ALLOWED-FOR-W]*W-M-L[ALLOWED-FOR-W]*)+
STRAIGHT-N-CLEAN	[N]
STRAIGHT-N-LENDRIGHT	[N-M][NNE-TO-E-NM]
STRAIGHT-N-LSTARTLEFT	[NNE-TO-E-NM][N-M]
STRAIGHT-N-LENDLEFT	[N-M][NNW-TO-W-NM]
STRAIGHT-N-LSTARTRIGHT	[NNW-TO-W-NM][N-M]
STRAIGHT-N-LLSTARTLEFT	[E-45-NM][N-M][E-45-NM]
STRAIGHT-N-LLSTARTRIGHT	[W-45-NM][N-M][W-45-NM]
STRAIGHT-N-U	([NNE-TO-E-NM][N-M][NNW-TO-W-NM])|([NNE-TO-NEE-NM][NNW-TO-NWW-NM])
STRAIGHT-N-N	([NNW-TO-W-NM][N-M][NNE-TO-E-NM])|([NNW-TO-NWW-NM][NNE-TO-NEE-NM])
STRAIGHT-N-Z	[N-M][NNE-TO-E-NM][N-M]
STRAIGHT-N-S	[N-M][NNW-TO-W-NM][N-M]
STRAIGHT-N-UNCLEAN	([ALLOWED-FOR-N]*N-M-L[ALLOWED-FOR-N]*)+
L-SE-CLEAN	SM EM | SNM ENM
L-SE-LENDRIGHT	SM EM [SEE-TO-S-NM]
L-SE-LSTARTLEFT	[SSW-TO-W-NM] SM EM
L-SE-LENDLEFT	SM EM [NEE-TO-N-NM]
L-SE-LSTARTRIGHT	[SSE-TO-E-NM] SM EM
L-SE-LLSTARTLEFT	[SSW-TO-W-NM] SM EM [SEE-TO-S-NM]
L-SE-LLSTARTRIGHT	[SSE-TO-E-NM] SM EM [NEE-TO-N-NM]
L-SE-U	[SSW-TO-W-NM] SM EM [NEE-TO-N-NM]
L-SE-N	[SSE-TO-E-NM] SM EM [SEE-TO-S-NM]
L-SE-UNCLEAN	([ALLOWED-FOR-S]*S-M-L[ALLOWED-FOR-S]*)+([ALLOWED-FOR-E]*E-M-L[ALLOWED-FOR-E]*)+
L-WS-CLEAN	WM SM | WNM SNM
L-WS-LENDRIGHT	WM SM [SSW-TO-W-NM]
L-WS-LSTARTLEFT	[NWW-TO-N-NM] WM SM
L-WS-LENDLEFT	WM SM [SSE-TO-E-NM]
L-WS-LSTARTRIGHT	[SWW-TO-S-NM] WM SM
L-WS-LLSTARTLEFT	[NWW-TO-N-NM] WM SM [SSW-TO-W-NM]
L-WS-LLSTARTRIGHT	[SWW-TO-S-NM] WM SM [SSE-TO-E-NM]
L-WS-U	[NWW-TO-N-NM] WM SM [SSE-TO-E-NM]
L-WS-N	[SWW-TO-S-NM] WM SM [SSW-TO-W-NM]
L-WS-UNCLEAN	([ALLOWED-FOR-W]*W-M-L[ALLOWED-FOR-W]*)+([ALLOWED-FOR-S]*S-M-L[ALLOWED-FOR-S]*)+

L-NW-CLEAN
NM WM | NNM WNM
L-NW-LENDRIGHT
NM WM | NWW-TO-N-NM
L-NW-LSTARTLEFT
NNE-TO-E-NM | NM WM
L-NW-LENDLEFT
NM WM | SWW-TO-S-NM
L-NW-LSTARTRIGHT
NNW-TO-W-NM | NM WM
L-NW-LLSTARTLEFT
NNE-TO-E-NM | NM WM | NWW-TO-N-NM
L-NW-LLSTARTRIGHT
NNW-TO-W-NM | NM WM | SWW-TO-S-NM
L-NW-U
NNE-TO-E-NM | NM WM | SWW-TO-S-NM
L-NW-N
NNW-TO-W-NM | NM WM | NWW-TO-N-NM
L-NW-UNCLEAN
(|ALLOWED-FOR-N| *N-M-L |ALLOWED-FOR-N| *)+(|ALLOWED-FOR-W| *W-M-L |ALLOWED-FOR-W| *)+
L-EN-CLEAN
EM NM | ENM NNM
L-EN-LENDRIGHT
EM NM | NNE-TO-E-NM
L-EN-LSTARTLEFT
SEE-TO-S-NM | EM NM
L-EN-LENDLEFT
EM NM | NNW-TO-W-NM
L-EN-LSTARTRIGHT
NEE-TO-N-NM | EM NM
L-EN-LLSTARTLEFT
SEE-TO-S-NM | EM NM | NNE-TO-E-NM
L-EN-LLSTARTRIGHT
NEE-TO-N-NM | EM NM | NNW-TO-W-NM
L-EN-U
SEE-TO-S-NM | EM NM | NNW-TO-W-NM
L-EN-N
NEE-TO-N-NM | EM NM | NNE-TO-E-NM
L-EN-UNCLEAN
(|ALLOWED-FOR-E| *E-M-L |ALLOWED-FOR-E| *)+(|ALLOWED-FOR-N| *N-M-L |ALLOWED-FOR-N| *)+
L-SW-CLEAN
SM WM | SNM WNM
L-SW-LENDRIGHT
SM WM | NWW-TO-N-NM
L-SW-LSTARTLEFT
SSW-TO-W-NM | SM WM
L-SW-LENDLEFT
SM WM | SWW-TO-S-NM
L-SW-LSTARTRIGHT
SSE-TO-E-NM | SM WM
L-SW-LLSTARTLEFT
SSW-TO-W-NM | SM WM | NWW-TO-N-NM
L-SW-LLSTARTRIGHT
SSE-TO-E-NM | SM WM | SWW-TO-S-NM
L-SW-U
SSE-TO-E-NM | SM WM | NWW-TO-N-NM
L-SW-N
SSW-TO-W-NM | SM WM | SWW-TO-S-NM
L-SW-UNCLEAN
(|ALLOWED-FOR-S| *S-M-L |ALLOWED-FOR-S| *)+(|ALLOWED-FOR-W| *W-M-L |ALLOWED-FOR-W| *)+
L-WN-CLEAN
WM NM | WNM NNM
L-WN-LENDRIGHT
WM NM | NNE-TO-E-NM
L-WN-LSTARTLEFT
NWW-TO-N-NM | WM NM

L-WN-LENDLEFT	WM|NM|[NNW-TO-W-NM]
L-WN-LSTARTRIGHT	[SWW-TO-S-NM]|WM|NM
L-WN-LLSTARTLEFT	[NWW-TO-N-NM]|WM|NM|[NNW-TO-W-NM]
L-WN-LLSTARTRIGHT	[SWW-TO-S-NM]|WM|NM|[NNE-TO-E-NM]
L-WN-U	[SWW-TO-S-NM]|WM|NM|[NNE-TO-E-NM]
L-WN-N	[NWW-TO-N-NM]|WM|NM|[NNW-TO-W-NM]
L-WN-UNCLEAN	([ALLOWED-FOR-W]*W-M-L[ALLOWED-FOR-W]*)+([ALLOWED-FOR-N]*N-M-L[ALLOWED-FOR-N]*)+
L-NE-CLEAN	NM|EM|NNM|ENM
L-NE-LENDRIGHT	NM|EM|[SEE-TO-S-NM]
L-NE-LSTARTLEFT	[NNE-TO-E-NM]|NM|EM
L-NE-LENDLEFT	NM|EM|[NEE-TO-N-NM]
L-NE-LSTARTRIGHT	[NNW-TO-W-NM]|NM|EM
L-NE-LLSTARTLEFT	[NNE-TO-E-NM]|NM|EM|[SEE-TO-S-NM]
L-NE-LLSTARTRIGHT	[NNW-TO-W-NM]|NM|EM|[NEE-TO-N-NM]
L-NE-U	[NNW-TO-W-NM]|NM|EM|[SEE-TO-S-NM]
L-NE-N	[NNE-TO-E-NM]|NM|EM|[NEE-TO-N-NM]
L-NE-UNCLEAN	([ALLOWED-FOR-N]*N-M-L[ALLOWED-FOR-N]*)+([ALLOWED-FOR-E]*E-M-L[ALLOWED-FOR-E]*)+
L-ES-CLEAN	EM|SM|ENM|SNM
L-ES-LENDRIGHT	EM|SM|[SSW-TO-W-NM]
L-ES-LSTARTLEFT	[SEE-TO-S-NM]|EM|SM
L-ES-LENDLEFT	EM|SM|[SSE-TO-E-NM]
L-ES-LSTARTRIGHT	[NEE-TO-N-NM]|EM|SM
L-ES-LLSTARTLEFT	[SEE-TO-S-NM]|EM|SM|[SSW-TO-W-NM]
L-ES-LLSTARTRIGHT	[NEE-TO-N-NM]|EM|SM|[SSE-TO-E-NM]
L-ES-U	[NEE-TO-N-NM]|EM|SM|[SSW-TO-W-NM]
L-ES-N	[SEE-TO-S-NM]|EM|SM|[SSE-TO-E-NM]
L-ES-UNCLEAN	([ALLOWED-FOR-E]*E-M-L[ALLOWED-FOR-E]*)+([ALLOWED-FOR-S]*S-M-L[ALLOWED-FOR-S]*)+
SNAKE-ES-CLEAN	([E-M]|[S-90-NM|SWW-NM]?[W-M]([S-90-NM|SEE-NM]?[E]|[S-90-NM|SWW-NM]?[W])*([S-90-NM|SEE-NM]?[E])? [S-90-NM|SWW-NM]?)|([E]|[S-90-NM|SWW-NM]|[W]|[S-90-NM|SEE-NM]|[E](([S-90-NM|SWW-NM]|[W]|[S-90-NM|SEE-NM]|[E])* ([S-90-NM|SWW-NM]|[W])?[S-90-NM|SEE-NM]?)
SNAKE-ES-STARTTOSIDE	[ALL-NM]|[E-M]|[S-90-NM|SWW-NM]?[W-M]([S-90-NM|SEE-NM]?[E]|[S-90-NM|SWW-NM]?[W])*([S-90-NM|SEE-NM]?[E])? [S-90-NM|SWW-NM]?
SNAKE-WS-CLEAN	([W-M]|[S-90-NM|SEE-NM]?[E-M]([S-90-NM|SWW-NM]?[W]|[S-90-NM|SEE-NM]?[E])*([S-90-NM|SWW-NM]?[W])? [S-90-NM|SEE-NM]?)|([W]|[S-90-NM|SEE-NM]|[E]|[S-90-NM|SWW-NM]|[W](([S-90-NM|SEE-NM]|[E]|[S-90-NM|SWW-NM]|[W])*

	([S-90-NM|SEE-NM||E|)?[S-90-NM|SWW-NM|?)
SNAKE-WS-STARTTOSIDE	[ALL-NM||W-M||S-90-NM|SEE-NM|]?[E-M]([S-90-NM|SWW-NM|]?[W]||S-90-NM|SEE-NM|]?[E])*([S-90-NM|SWW-NM|]?[W])? [S-90-NM|SEE-NM|]?
SNAKE-WN-CLEAN	([W-M]||N-90-NM|NEE-NM|]?[E-M]([N-90-NM|NWW-NM|]?[W]||N-90-NM|NEE-NM|]?[E])*([N-90-NM|NWW-NM|]?[W])? [N-90-NM|NEE-NM|?)([W]||N-90-NM|NEE-NM|]||E]||N-90-NM|NWW-NM] [W]([N-90-NM|NEE-NM|]||E]||N-90-NM|NWW-NM|]||W])*([N-90-NM|NEE-NM|]||E])?[N-90-NM|NWW-NM|?)
SNAKE-WN-STARTTOSIDE	[ALL-NM||W-M||N-90-NM|NEE-NM|]?[E-M]([N-90-NM|NWW-NM|]?[W]||N-90-NM|NEE-NM|]?[E])*([N-90-NM|NWW-NM|]? [W])?[N-90-NM|NEE-NM|]?
SNAKE-EN-CLEAN	([E-M]||N-90-NM|NWW-NM|?||W-M]([N-90-NM|NEE-NM|]?[E]||N-90-NM|NWW-NM|]?[W])*([N-90-NM|NEE-NM|]?[E])? [N-90-NM|NWW-NM|?)|([E]||N-90-NM|NWW-NM|]||W]||N-90-NM|NEE-NM|]||E])([N-90-NM|NWW-NM|]||W]||N-90-NM|NEE-NM]
SNAKE-EN-STARTTOSIDE	[E])*([N-90-NM|NWW-NM|]||W])?[N-90-NM|NEE-NM|?) [ALL-NM||E-M]||N-90-NM|NWW-NM|]?[W-M]([N-90-NM|NEE-NM|]?[E]||N-90-NM|NWW-NM|]?[W])*([N-90-NM|NEE-NM|]?
SNAKE-SE-CLEAN	[E])?[N-90-NM|NWW-NM|]? ([S-M]||E-90-NM|NNE-NM|]?[N-M]([E-90-NM|SSE-NM|]?[S]||E-90-NM|NNE-NM|]?[N])*([E-90-NM|SSE-NM|]?[S])? [E-90-NM|NNE-NM|?)([S]||E-90-NM|NNE-NM|]||N]||E-90-NM|SSE-NM|]||S])([E-90-NM|NNE-NM|]||N])?[E-90-NM|SSE-NM|?)*
SNAKE-SE-STARTTOSIDE	([E-90-NM|NNE-NM|?) [ALL-NM||S-M]||E-90-NM|NNE-NM|]?[N-M]([E-90-NM|SSE-NM|]?[S]||E-90-NM|NNE-NM|]?[N])*([E-90-NM|SSE-NM|]?[S])? [E-90-NM|NNE-NM|]?
SNAKE-NE-CLEAN	([N-M]||E-90-NM|SSE-NM|]?[S-M]([E-90-NM|NNE-NM|]?[N]||E-90-NM|SSE-NM|]?[S])*([E-90-NM|NNE-NM|]?[N])? [E-90-NM|SSE-NM|?)|([N]||E-90-NM|SSE-NM|]||S]||E-90-NM|NNE-NM|]||N])([E-90-NM|SSE-NM|]||S]||E-90-NM|NNE-NM|])*
SNAKE-NE-STARTTOSIDE	([E-90-NM|SSE-NM|]||S])?[E-90-NM|NNE-NM|?) [ALL-NM||N-M]||E-90-NM|SSE-NM|]?[S-M]([E-90-NM|NNE-NM|]?[N]||E-90-NM|SSE-NM|]?[S])*([E-90-NM|NNE-NM|]?[N])? [E-90-NM|SSE-NM|]?
SNAKE-NW-CLEAN	([N-M]||W-90-NM|SSW-NM|]?[S-M]([W-90-NM|NNW-NM|]?[N]||W-90-NM|SSW-NM|]?[S])*([W-90-NM|NNW-NM|]?[N])? [W-90-NM|SSW-NM|?)([N]||W-90-NM|SSW-NM|]||S])?[W-90-NM|NNW-NM|?) [N])*([W-90-NM|SSW-NM|]?)
SNAKE-NW-STARTTOSIDE	[ALL-NM||N-M]||W-90-NM|SSW-NM|]?[S-M]([W-90-NM|NNW-NM|]?[N]||W-90-NM|SSW-NM|]?[S])*([W-90-NM|NNW-NM|]?
SNAKE-SW-CLEAN	[N])?[W-90-NM|SSW-NM|]? ([S-M]||W-90-NM|NNW-NM|]?[N-M]([W-90-NM|SSW-NM|]?[S]||W-90-NM|NNW-NM|]?[N])*([W-90-NM|SSW-NM|]?[S])?
SNAKE-SW-STARTTOSIDE	W-90-NM|NNW-NM|?)|([S]||W-90-NM|NNW-NM|]||N])?[W-90-NM|SSW-NM|]||S])([W-90-NM|NNW-NM|]||N])([W-90-NM|SSW-NM] [S])*([W-90-NM|NNW-NM|?) [ALL-NM||S-M]||W-90-NM|NNW-NM|]?[N-M]([W-90-NM|SSW-NM|]?[S]||W-90-NM|NNW-NM|]?[N])*([W-90-NM|SSW-NM|]?
MULTILINE-ES-CLEAN	S])?[W-90-NM|NNW-NM|?
MULTILINE-ES-STARTTOSIDE	[E]||SSW-TO-W-NM-L]||E]||SSW-TO-W-NM-L]||E])*
MULTILINE-ES-ENDTOSIDE	[ALL-NM]||E]||SSW-TO-W-NM-L]||E]||SSW-TO-W-NM-L]||E])* [E]||SSW-TO-W-NM-L]||E]||SSW-TO-W-NM-L]||E])*[ALL-NM]

MULTILINE-ES-STARTANDENDTOSIDE	[ALL-NM][E]([SSW-TO-W-NM-L][E])*[ALL-NM]
MULTILINE-WN-CLEAN	[W][NNE-TO-E-NM-L][W]([NNE-TO-E-NM-L][W])*
MULTILINE-WN-STARTTOSIDE	[ALL-NM][W][NNE-TO-E-NM-L][W]([NNE-TO-E-NM-L][W])*
MULTILINE-WN-ENDTOSIDE	[W][NNE-TO-E-NM-L][W]([NNE-TO-E-NM-L][W])*[ALL-NM]
MULTILINE-WN-STARTANDENDTOSIDE	[ALL-NM][W][NNE-TO-E-NM-L][W]([NNE-TO-E-NM-L][W])*[ALL-NM]
MULTILINE-EN-CLEAN	[E][NNW-TO-W-NM-L][E]([NNW-TO-W-NM-L][E])*
MULTILINE-EN-STARTTOSIDE	[ALL-NM][E][NNW-TO-W-NM-L][E]([NNW-TO-W-NM-L][E])*
MULTILINE-EN-ENDTOSIDE	[E][NNW-TO-W-NM-L][E]([NNW-TO-W-NM-L][E])*[ALL-NM]
MULTILINE-EN-STARTANDENDTOSIDE	[ALL-NM][E][NNW-TO-W-NM-L][E]([NNW-TO-W-NM-L][E])*[ALL-NM]
MULTILINE-WS-CLEAN	[W][SSE-TO-E-NM-L][W]([SSE-TO-E-NM-L][W])*
MULTILINE-WS-STARTTOSIDE	[ALL-NM][W][SSE-TO-E-NM-L][W]([SSE-TO-E-NM-L][W])*
MULTILINE-WS-ENDTOSIDE	[W][SSE-TO-E-NM-L][W]([SSE-TO-E-NM-L][W])*[ALL-NM]
MULTILINE-WS-STARTANDENDTOSIDE	[ALL-NM][W][SSE-TO-E-NM-L][W]([SSE-TO-E-NM-L][W])*[ALL-NM]
MULTILINE-SE-CLEAN	[S][NEE-TO-N-NM-L][S]([NEE-TO-N-NM-L][S])*
MULTILINE-SE-STARTTOSIDE	[ALL-NM][S][NEE-TO-N-NM-L][S]([NEE-TO-N-NM-L][S])*
MULTILINE-SE-ENDTOSIDE	[S][NEE-TO-N-NM-L][S]([NEE-TO-N-NM-L][S])*[ALL-NM]
MULTILINE-SE-STARTANDENDTOSIDE	[ALL-NM][S][NEE-TO-N-NM-L][S]([NEE-TO-N-NM-L][S])*[ALL-NM]
MULTILINE-NW-CLEAN	[N][SWW-TO-S-NM-L][N]([SWW-TO-S-NM-L][N])*
MULTILINE-NW-STARTTOSIDE	[ALL-NM][N][SWW-TO-S-NM-L][N]([SWW-TO-S-NM-L][N])*
MULTILINE-NW-ENDTOSIDE	[N][SWW-TO-S-NM-L][N]([SWW-TO-S-NM-L][N])*[ALL-NM]
MULTILINE-NW-STARTANDENDTOSIDE	[ALL-NM][N][SWW-TO-S-NM-L][N]([SWW-TO-S-NM-L][N])*[ALL-NM]
MULTILINE-NE-CLEAN	[S][SEE-TO-S-NM-L][S]([SEE-TO-S-NM-L][S])*
MULTILINE-NE-STARTTOSIDE	[ALL-NM][S][SEE-TO-S-NM-L][S]([SEE-TO-S-NM-L][S])*
MULTILINE-NE-ENDTOSIDE	[S][SEE-TO-S-NM-L][S]([SEE-TO-S-NM-L][S])*[ALL-NM]
MULTILINE-NE-STARTANDENDTOSIDE	[ALL-NM][S][SEE-TO-S-NM-L][S]([SEE-TO-S-NM-L][S])*[ALL-NM]
MULTILINE-SW-CLEAN	[N][NWW-TO-N-NM-L][N]([NWW-TO-N-NM-L][N])*
MULTILINE-SW-STARTTOSIDE	[ALL-NM][N][NWW-TO-N-NM-L][N]([NWW-TO-N-NM-L][N])*
MULTILINE-SW-ENDTOSIDE	[N][NWW-TO-N-NM-L][N]([NWW-TO-N-NM-L][N])*[ALL-NM]
MULTILINE-SL-STARTANDENDTOSIDE	[ALL-NM][N][NWW-TO-N-NM-L][N]([NWW-TO-N-NM-L][N])*[ALL-NM]

U-SE	([SEE-TO-S-M][E-M][NEE-TO-N-M])|([SEE-TO-S-NM][E-NM][NEE-TO-N-NM])
U-WS	([SSW-TO-W-M][S-M][SSE-TO-E-M])|([SSW-TO-W-NM][S-NM][SSE-TO-E-NM])
U-NW	([NWW-TO-N-M][W-M][SWW-TO-S-M])|([NWW-TO-N-NM][W-NM][SWW-TO-S-NM])
U-EN	([NNE-TO-E-M][N-M][NNW-TO-W-M])|([NNE-TO-E-NM][N-NM][NNW-TO-W-NM])
U-SW	([SWW-TO-S-M][W-M][NWW-TO-N-M])|([SWW-TO-S-NM][W-NM][NWW-TO-N-NM])
U-WN	([NNW-TO-W-M][N-M][NNE-TO-E-M])|([NNW-TO-W-NM][N-NM][NNE-TO-E-NM])
U-NE	([NEE-TO-N-M][E-M][SEE-TO-S-M])|([NEE-TO-N-NM][E-NM][SEE-TO-S-NM])

U-ES

(([SSE-TO-E-M][S-M][SSW-TO-W-M])|([SSE-TO-E-NM][S-NM][SSW-TO-W-NM]))

STAIRS-SE

([S-NM-S[E-NM-S]?[SE-45]+[S-NM-S[E-NM-S]?)|([S-NM-L[E-NM-L([S-NM-L[E-NM-L])+)|
([E-NM-L[S-NM-L([E-NM-L[S-NM-L])+)

STAIRS-SW

([S-NM-S[W-NM-S]?[SW-45]+[S-NM-S[W-NM-S]?)|([S-NM-L[W-NM-L([S-NM-L[W-NM-L])+)|
([W-NM-L[S-NM-L([W-NM-L[S-NM-L])+)

STAIRS-NW

([N-NM-S[W-NM-S]?[NW-45]+[N-NM-S[W-NM-S]?)|([N-NM-L[W-NM-L([N-NM-L[W-NM-L])+)|
([W-NM-L[N-NM-L([W-NM-L[N-NM-L])+)

STAIRS-NE

([N-NM-S[E-NM-S]?[NE-45]+[N-NM-S[E-NM-S]?)|([N-NM-L[E-NM-L([N-NM-L[E-NM-L])+)|
([E-NM-L[N-NM-L([E-NM-L[N-NM-L])+)

Z-ES-CLEAN

EM|SM|EM|[]|ENM|SNM|ENM

Z-ES-LENDRIGHT

EM|SM|EM|[SEE-TO-S-NM]

Z-ES-LSTARTLEFT

[SEE-TO-S-NM]|EM|SM|EM

Z-ES-LENDLEFT

EM|SM|EM|[NEE-TO-N-NM]

Z-ES-LSTARTRIGHT

[NEE-TO-N-NM]|EM|SM|EM

Z-ES-LLSTARTLEFT

[SEE-TO-S-NM]|EM|SM|EM|[SEE-TO-S-NM]

Z-ES-LLSTARTRIGHT

[NEE-TO-N-NM]|EM|SM|EM|[NEE-TO-N-NM]

Z-ES-U

[SEE-TO-S-NM]|EM|SM|EM|[NEE-TO-N-NM]

Z-ES-N

[NEE-TO-N-NM]|EM|SM|EM|[SEE-TO-S-NM]

Z-ES-UNCLEAN

([ALLOWED-FOR-E]|*[E-M-L][ALLOWED-FOR-E]*)+([ALLOWED-FOR-S]|*[S-M-L][ALLOWED-FOR-S]*)+
([ALLOWED-FOR-E]|*[E-M-L][ALLOWED-FOR-E]*)+

Z-EN-CLEAN

EM|NM|EM|[]|ENM|NNM|ENM

Z-EN-LENDRIGHT

EM|NM|EM|[SEE-TO-S-NM]

Z-EN-LSTARTLEFT

[SEE-TO-S-NM]|EM|NM|EM

Z-EN-LENDLEFT

EM|NM|EM|[NEE-TO-N-NM]

Z-EN-LSTARTRIGHT

[NEE-TO-N-NM]|EM|NM|EM

Z-EN-LLSTARTLEFT

[SEE-TO-S-NM]|EM|NM|EM|[SEE-TO-S-NM]

Z-EN-LLSTARTRIGHT

[NEE-TO-N-NM]|EM|NM|EM|[NEE-TO-N-NM]

Z-EN-U

[SEE-TO-S-NM]|EM|NM|EM|[NEE-TO-N-NM]

Z-EN-N

[NEE-TO-N-NM]|EM|NM|EM|[SEE-TO-S-NM]

Z-EN-UNCLEAN

([ALLOWED-FOR-E]|*[E-M-L][ALLOWED-FOR-E]*)+([ALLOWED-FOR-N]|*[N-M-L][ALLOWED-FOR-N]*)+
([ALLOWED-FOR-E]|*[E-M-L][ALLOWED-FOR-E]*)+

Z-WS-CLEAN

WM|SM|WM|[]|WNM|SNM|WNM

Z-WS-LENDRIGHT

WM|SM|WM|[NWW-TO-N-NM]

Z-WS-LSTARTLEFT	[NWW-TO-N-NM][WM][SM][WM]
Z-WS-LENDLEFT	[WM][SM][WM][SWW-TO-S-NM]
Z-WS-LSTARTRIGHT	[SWW-TO-S-NM][WM][SM][WM]
Z-WS-LLSTARTLEFT	[NWW-TO-N-NM][WM][SM][WM][NWW-TO-N-NM]
Z-WS-LLSTARTRIGHT	[SWW-TO-S-NM][WM][SM][WM][SWW-TO-S-NM]
Z-WS-U	[NWW-TO-N-NM][WM][SM][WM][SWW-TO-S-NM]
Z-WS-N	[SWW-TO-S-NM][WM][SM][WM][NWW-TO-N-NM]
Z-WS-UNCLEAN	([ALLOWED-FOR-W]*[W-M-L][ALLOWED-FOR-W]*)+([ALLOWED-FOR-S]*[S-M-L][ALLOWED-FOR-S]*)+ ([ALLOWED-FOR-W]*[W-M-L][ALLOWED-FOR-W]*)+
Z-WN-CLEAN	[WM][NM][WM][]WNM[NNM][WNM]
Z-WN-LENDRIGHT	[WM][NM][WM][NWW-TO-N-NM]
Z-WN-LSTARTLEFT	[NWW-TO-N-NM][WM][NM][WM]
Z-WN-LENDLEFT	[WM][NM][WM][SWW-TO-S-NM]
Z-WN-LSTARTRIGHT	[SWW-TO-S-NM][WM][NM][WM]
Z-WN-LLSTARTLEFT	[NWW-TO-N-NM][WM][NM][WM][NWW-TO-N-NM]
Z-WN-LLSTARTRIGHT	[SWW-TO-S-NM][WM][NM][WM][SWW-TO-S-NM]
Z-WN-U	[NWW-TO-N-NM][WM][NM][WM][SWW-TO-S-NM]
Z-WN-N	[SWW-TO-S-NM][WM][NM][WM][NWW-TO-N-NM]
Z-WN-UNCLEAN	([ALLOWED-FOR-W]*[W-M-L][ALLOWED-FOR-W]*)+([ALLOWED-FOR-N]*[N-M-L][ALLOWED-FOR-N]*)+ ([ALLOWED-FOR-W]*[W-M-L][ALLOWED-FOR-W]*)+
Z-SE-CLEAN	[SM][EM][SM][]SNM[ENM][SNM]
Z-SE-LENDRIGHT	[SM][EM][SM][SSW-TO-W-NM]
Z-SE-LSTARTLEFT	[SSW-TO-W-NM][SM][EM][SM]
Z-SE-LENDLEFT	[SM][EM][SM][SSE-TO-E-NM]
Z-SE-LSTARTRIGHT	[SSE-TO-E-NM][SM][EM][SM]
Z-SE-LLSTARTLEFT	[SSW-TO-W-NM][SM][EM][SM][SSW-TO-W-NM]
Z-SE-LLSTARTRIGHT	[SSE-TO-E-NM][SM][EM][SM][SSE-TO-E-NM]
Z-SE-U	[SSW-TO-W-NM][SM][EM][SM][SSE-TO-E-NM]
Z-SE-N	[SSE-TO-E-NM][SM][EM][SM][SSW-TO-W-NM]
Z-SE-UNCLEAN	([ALLOWED-FOR-S]*[S-M-L][ALLOWED-FOR-S]*)+([ALLOWED-FOR-E]*[E-M-L][ALLOWED-FOR-E]*)+ ([ALLOWED-FOR-S]*[S-M-L][ALLOWED-FOR-S]*)+
Z-SW-CLEAN	[SM][WM][SM][]SNM[WNM][SNM]
Z-SW-LENDRIGHT	[SM][WM][SM][SSW-TO-W-NM]
Z-SW-LSTARTLEFT	[SSW-TO-W-NM][SM][WM][SM]

Z-SW-LENDLEFT
SM|WM|SM|SSE-TO-E-NM
Z-SW-LSTARTRIGHT
SSE-TO-E-NM|SM|WM|SM
Z-SW-LLSTARTLEFT
SSW-TO-W-NM|SM|WM|SM|SSW-TO-W-NM
Z-SW-LLSTARTRIGHT
SSE-TO-E-NM|SM|WM|SM|SSE-TO-E-NM
Z-SW-U
SSW-TO-W-NM|SM|WM|SM|SSE-TO-E-NM
Z-SW-N
SSE-TO-E-NM|SM|WM|SM|SSW-TO-W-NM
Z-SW-UNCLEAN
([ALLOWED-FOR-S]*S-M-L[ALLOWED-FOR-S]*)+([ALLOWED-FOR-W]*W-M-L[ALLOWED-FOR-W]*)+
([ALLOWED-FOR-S]*S-M-L[ALLOWED-FOR-S]*)+
Z-NE-CLEAN
NM|EM|NM|[]|NNM|ENM|NNM
Z-NE-LENDRIGHT
NM|EM|NM|NNE-TO-E-NM
Z-NE-LSTARTLEFT
NNE-TO-E-NM|NM|EM|NM
Z-NE-LENDLEFT
NM|EM|NM|NNW-TO-W-NM
Z-NE-LSTARTRIGHT
NNW-TO-W-NM|NM|EM|NM
Z-NE-LLSTARTLEFT
NNE-TO-E-NM|NM|EM|NM|NNE-TO-E-NM
Z-NE-LLSTARTRIGHT
NNW-TO-W-NM|NM|EM|NM|NNW-TO-W-NM
Z-NE-U
NNE-TO-E-NM|NM|EM|NM|NNW-TO-W-NM
Z-NE-N
NNW-TO-W-NM|NM|EM|NM|NNE-TO-E-NM
Z-NE-UNCLEAN
([ALLOWED-FOR-N]*N-M-L[ALLOWED-FOR-N]*)+([ALLOWED-FOR-E]*E-M-L[ALLOWED-FOR-E]*)+
([ALLOWED-FOR-N]*N-M-L[ALLOWED-FOR-N]*)+
Z-NW-CLEAN
NM|WM|NM|[]|NNM|WNM|NNM
Z-NW-LENDRIGHT
NM|WM|NM|NNE-TO-E-NM
Z-NW-LSTARTLEFT
NNE-TO-E-NM|NM|WM|NM
Z-NW-LENDLEFT
NM|WM|NM|NNW-TO-W-NM
Z-NW-LSTARTRIGHT
NNW-TO-W-NM|NM|WM|NM
Z-NW-LLSTARTLEFT
NNE-TO-E-NM|NM|WM|NM|NNE-TO-E-NM
Z-NW-LLSTARTRIGHT
NNW-TO-W-NM|NM|WM|NM|NNW-TO-W-NM
Z-NW-U
NNE-TO-E-NM|NM|WM|NM|NNW-TO-W-NM
Z-NW-N
NNW-TO-W-NM|NM|WM|NM|NNE-TO-E-NM
Z-NW-UNCLEAN
([ALLOWED-FOR-N]*N-M-L[ALLOWED-FOR-N]*)+([ALLOWED-FOR-W]*W-M-L[ALLOWED-FOR-W]*)+
([ALLOWED-FOR-N]*N-M-L[ALLOWED-FOR-N]*)+

Anhang B

Verteilung der Fluss-Layouts

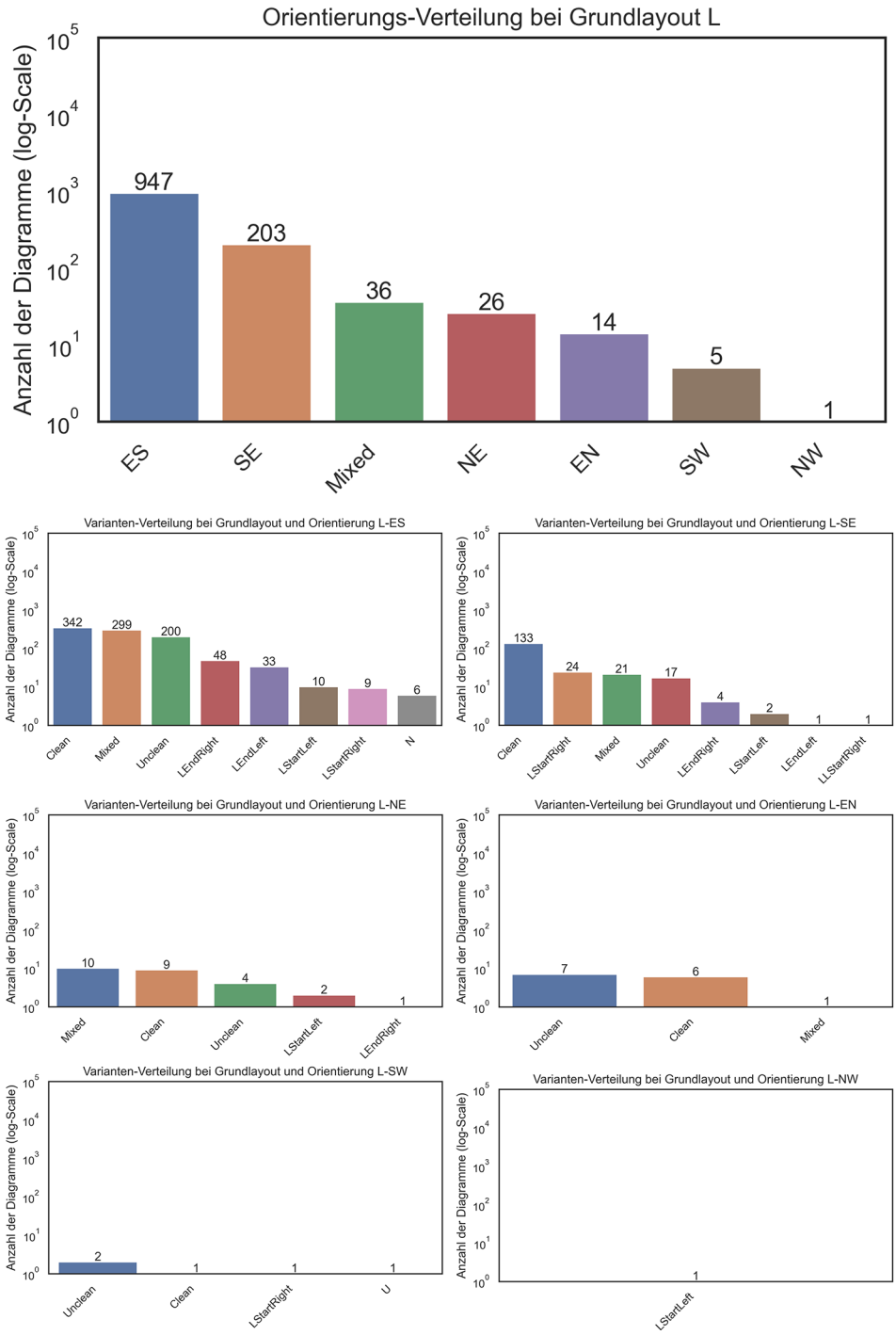

Abbildung B.1: Verteilung der Fluss-Layouts bei Grundlayout L

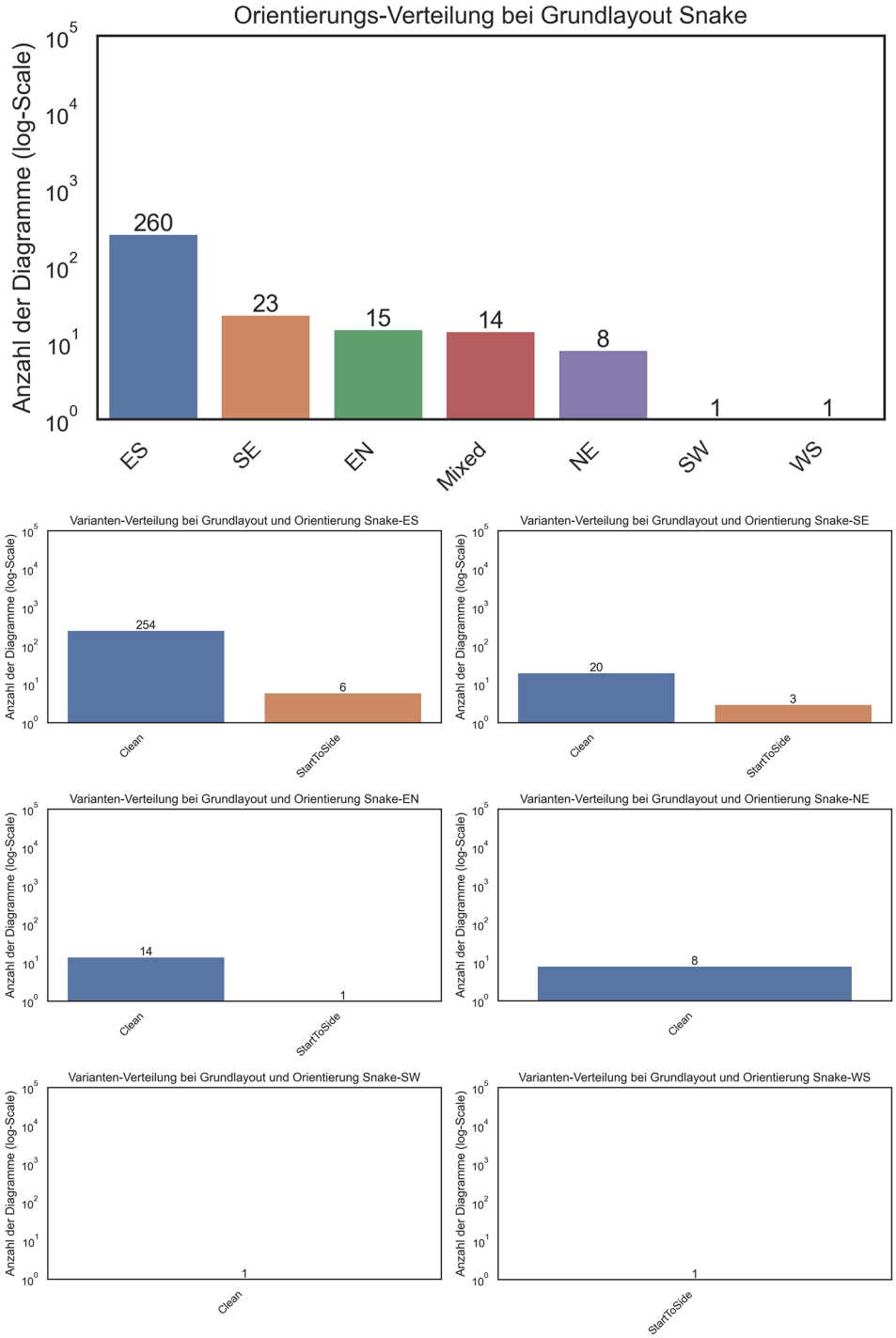

Abbildung B.2: Verteilung der Fluss-Layouts bei Grundlayout Snake

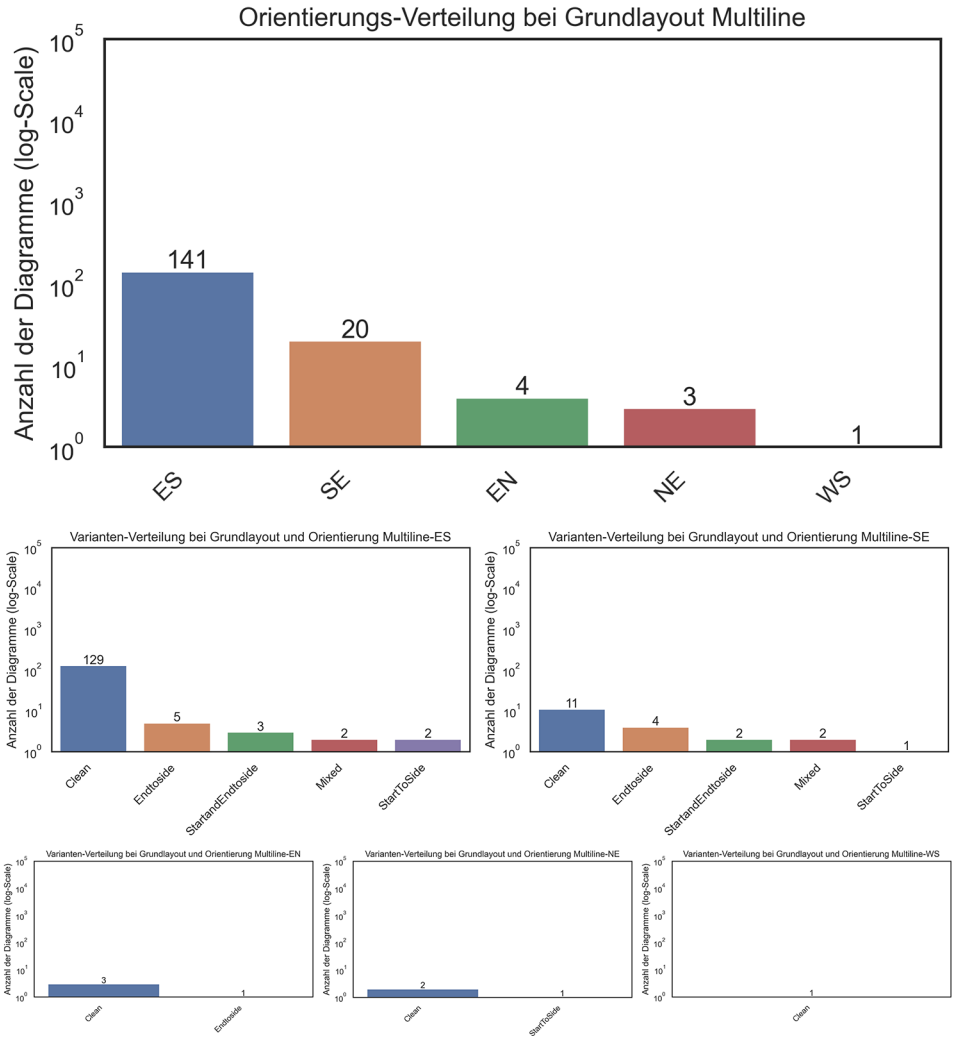

Abbildung B.3: Verteilung der Fluss-Layouts bei Grundlayout Multiline

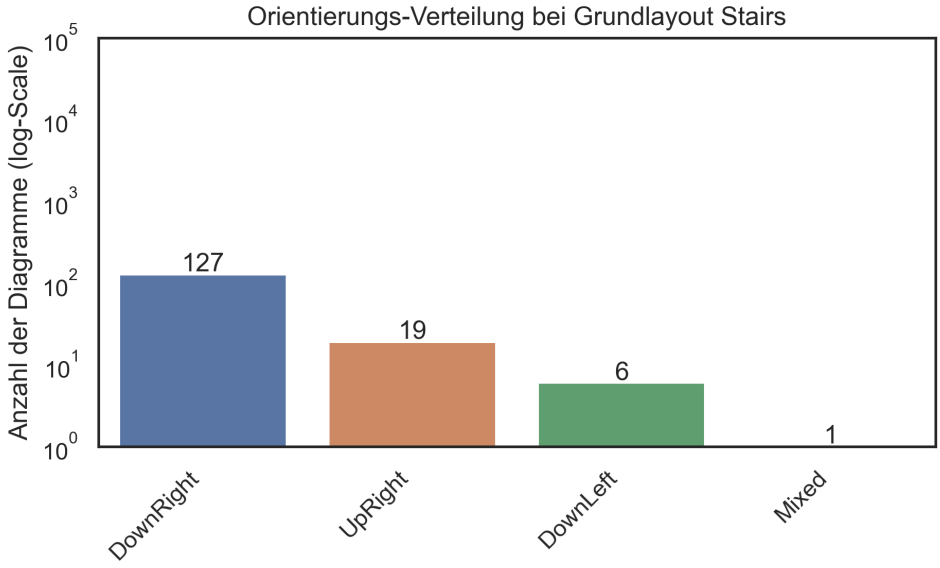

Abbildung B.4: Verteilung der Fluss-Layouts bei Grundlayout Stairs

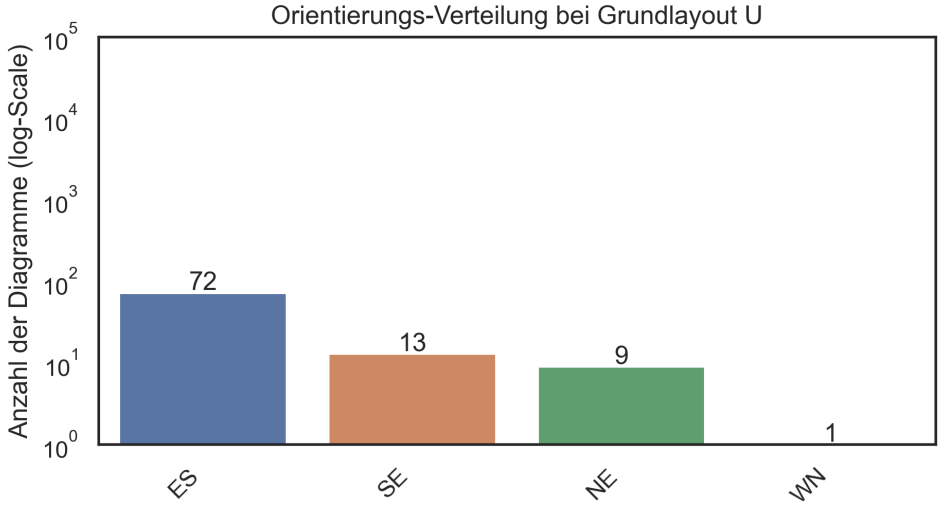

Abbildung B.5: Verteilung der Fluss-Layouts bei Grundlayout U

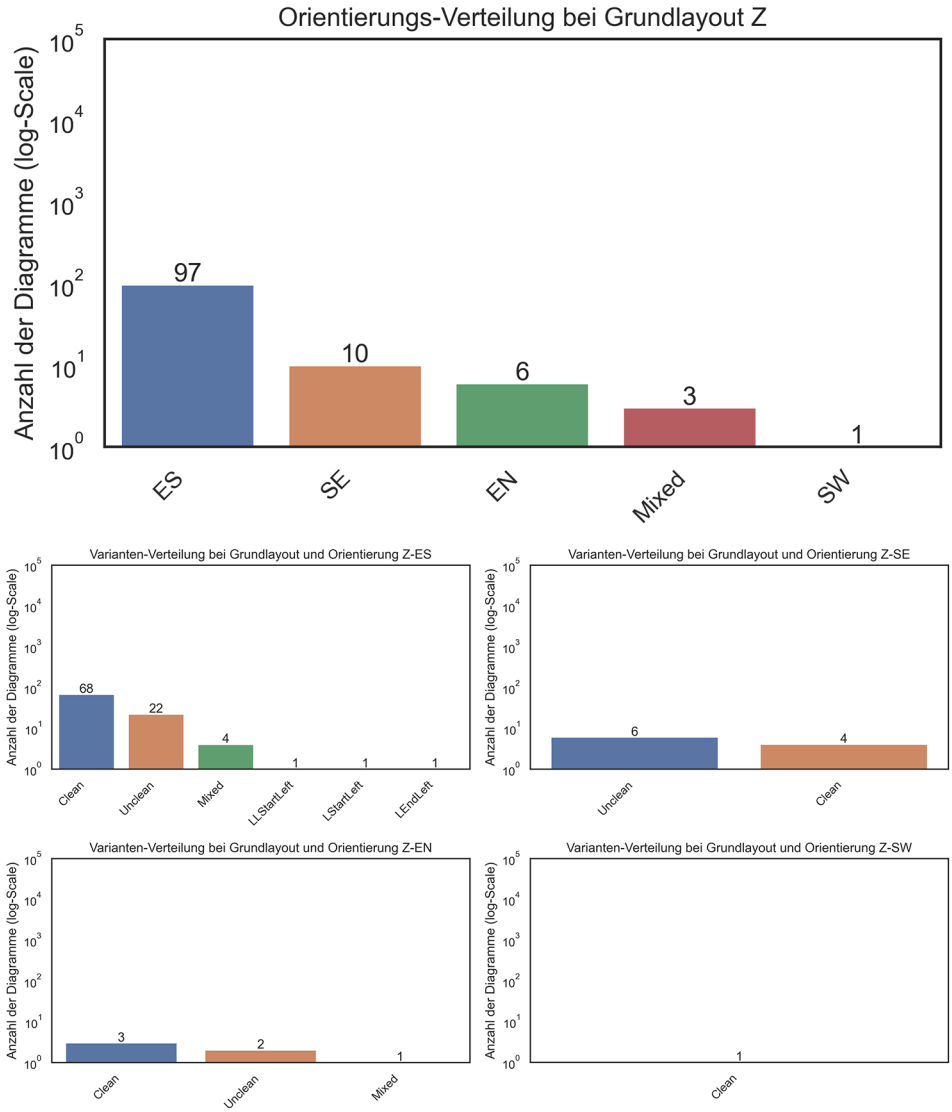

Abbildung B.6: Verteilung der Fluss-Layouts bei Grundlayout Z

Literaturverzeichnis

[1] Thomas Allweyer. *BPMN 2.0: Introduction to the standard for business process modeling*. 2nd, updated and extended edition. Norderstedt: BOD - Books on Demand, 2016. ISBN: 383709331X.

[2] Elias Baalmann. *bpmnflowlayoutclassifier: Automatic Flow-Layout Classification for BPMN-Diagrams*. 2022. URL: `https://github.com/baalmael/bpmnflowlayoutclassifier` (besucht am 01. 04. 2022).

[3] Elias Baalmann und Daniel Lübke. „Algorithmic Classification of Layouts of BPMN Diagrams“. In: *Proceedings of the 14th Central European Workshop on Services and their Composition (ZEUS 2022)*. Hrsg. von CEUR Workshop Proceedings. Bd. 3113. 2022, S. 42–50.

[4] Vered Bernstein und Pnina Soffer. „How Does It Look? Exploring Meaningful Layout Features of Process Models“. In: *Advanced Information Systems Engineering Workshops*. Hrsg. von Anne Persson und Janis Stirna. Cham: Springer International Publishing, 2015, S. 81–86. ISBN: 978-3-319-19243-7.

[5] BPM Offensive Berlin, Hrsg. *BPMN Poster*. 2011. URL: `http://www.bpmb.de/index.php/BPMNPoster` (besucht am 22. 02. 2022).

[6] BPMN.iO. *bpmn-to-image: Convert BPMN 2.0 diagrams to PDF documents or PNG files*. 2022. URL: `https://github.com/bpmn-io/bpmn-to-image` (besucht am 10. 03. 2022).

[7] Michele Chinosi und Alberto Trombetta. „BPMN: An introduction to the standard“. In: *Computer Standards & Interfaces* 34.1 (2012), S. 124–134. ISSN: 09205489. DOI: `10.1016/j.csi.2011.06.002`.

[8] Flavio Corradini u. a. „A Guidelines framework for understandable BPMN models“. In: *Data & Knowledge Engineering* 113 (2018), S. 129–154. ISSN: 0169023X. DOI: `10.1016/j.datak.2017.11.003`.

[9] Flavio Corradini u. a. *Quality assessment strategy: applying business process understandability guidelines for learning*. Hrsg. von Tech. Rep. 4.1. Italien, 2015. URL: `http://pumax.isti.cnr.it/dfdownloadnew.php?ident=cnr.isti/cnr.isti/2015-TR-034&langver=it&scelta=Metadata` (besucht am 06. 12. 2021).

[10] Philip Effinger. „Layout Patterns with BPMN Semantics“. In: *Business Process Model and Notation*. Hrsg. von Remco Dijkman, Jörg Hofstetter und Jana Koehler. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 130–135. ISBN: 978-3-642-25160-3. DOI: 10.1007/978-3-642-25160-3_11.

[11] Philip Effinger, Nicole Jogsch und Sandra Seiz. „On a Study of Layout Aesthetics for Business Process Models Using BPMN“. In: *Business Process Modeling Notation*. Hrsg. von Jan Mendling, Matthias Weidlich und Mathias Weske. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, S. 31–45. ISBN: 978-3-642-16298-5. DOI: 10.1007/978-3-642-16298-5_5.

[12] Philip Effinger, Martin Siebenhaller und Michael Kaufmann. „An Interactive Layout Tool for BPMN“. In: *2009 IEEE Conference on Commerce and Enterprise Computing*. Hrsg. von IEEE Computer Society. Wien, 2009, S. 399–406. DOI: 10.1109/CEC.2009.36.

[13] Holger Eichelberger und Klaus Schmid. „Guidelines on the aesthetic quality of UML class diagrams“. In: *Information and Software Technology* 51.12 (2009), S. 1686–1698. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2009.04.008.

[14] Kathrin Figl und Mark Strembeck. „Findings from an experiment on flow direction of business process models“. In: *Enterprise modelling and information systems architectures*. Hrsg. von Jens Kolb, Henrik Leopold und Jan Mendling. Bonn: Gesellschaft für Informatik e.V, 2015, S. 59–73. ISBN: 978-3-88579-642-8.

[15] Kathrin Figl und Mark Strembeck. „On the importance of flow direction in business process models“. In: *9th International Conference on Software Engineering and Applications (ICSOFT-EA)*. Hrsg. von IEEE Computer Society. Los Alamitos, CA, USA, 2014, S. 132–136. DOI: 10.13140/2.1.3445.8247.

[16] Jakob Freund und Bernd Rücker. *Praxishandbuch BPMN 2.0*. 4., aktual. Aufl. München: Hanser, 2014. ISBN: 9783446442924.

[17] Yoshimi Harsel und Roger Wales. „Directional Preference in Problem Solving“. In: *International Journal of Psychology* 22.2 (1987), S. 195–206. ISSN: 0020-7594. DOI: 10.1080/00207598708246777.

[18] Thomas Heinze, Viktor Stefanko und Wolfram Amme. „BPMN in the Wild: BPMN on GitHub.com“. In: *Proceedings of the 12th ZEUS Workshop on Services and their Composition*. Hrsg. von CEUR Workshop Proceedings. CEUR-WS.org, 2020, S. 26–29.

[19] Moritz Hesse. *BPMN Tool Matrix*. 2021. URL: https://bpmnmatrix.github.io/ (besucht am 19. 03. 2022).

[20] Eirini Kalliamvakou u. a. „The promises and perils of mining GitHub“. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. Hrsg. von Association for Computing Machinery. New York, NY: ACM, 2014, S. 92–101. ISBN: 9781450328630. DOI: 10.1145/2597073.2597074.

[21] Ingo Kitzmann u. a. „A Simple Algorithm for Automatic Layout of BPMN Processes“. In: *2009 IEEE Conference on Commerce and Enterprise Computing*. Hrsg. von IEEE Computer Society. Wien, 2009, S. 391–398. DOI: 10.1109/CEC.2009.28.

[22] Matthias Kurz, Falko Menge und Zbigniew Misiak. *Diagram Interchangeability in BPMN 2*. 2014. URL: https://www.omg.org/oceb-2/documents/BPMN_Interchange.pdf (besucht am 08. 10. 2021).

[23] Daniel Lübke. *bpmnlayoutanalyzer: Metric Tool for BPMN Diagrams*. 2022. URL: https://github.com/dluebke/bpmnlayoutanalyzer/ (besucht am 11. 03. 2022).

[24] Daniel Lübke, Maike Ahrens und Kurt Schneider. „Influence of diagram layout and scrolling on understandability of BPMN processes: an eye tracking experiment with BPMN diagrams“. In: *Information Technology and Management* 22.2 (2021), S. 99–131. ISSN: 1385-951X. DOI: 10.1007/s10799-021-00327-7.

[25] Daniel Lübke und Daniel Wutke. „Analysis of Prevalent BPMN Layout Choices on GitHub“. In: *Proceedings of the 13th European Workshop on Services and their Composition (ZEUS 2021)*. Hrsg. von CEUR Workshop Proceedings. Bd. 2839. 2021, S. 46–54.

[26] Jan Mendling, Hajo A. Reijers und Jorge Cardoso. „What Makes Process Models Understandable?“ In: *Business process management*. Hrsg. von Springer. Bd. 4714. Lecture Notes in Computer Science. Berlin und Heidelberg, 2007, S. 48–63. ISBN: 978-3-540-75182-3. DOI: 10.1007/978-3-540-75183-0_4.

[27] Joan C. Nordbotten und Martha E. Crosby. „The effect of graphic style on data model interpretation“. In: *Information Systems Journal* 9.2 (1999), S. 139–155. ISSN: 1350-1917. DOI: 10.1046/j.1365-2575.1999.00052.x.

[28] Object Management Group. *Business Process Model and Notation (BPMN), Version 1.2*. URL: https://www.omg.org/spec/BPMN/1.2/PDF (besucht am 08. 10. 2021).

[29] Object Management Group. *Business Process Model and Notation (BPMN), Version 2.0*. URL: https://www.omg.org/spec/BPMN/2.0/PDF (besucht am 08. 10. 2021).

[30] Oracle, Hrsg. *Pattern (Java Platform SE 7)*. 2022. URL: https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html (besucht am 29. 01. 2022).

[31] Helen C. Purchase. „Metrics for Graph Drawing Aesthetics“. In: *Journal of Visual Languages & Computing* 13.5 (2002), S. 501–516. ISSN: 1045926X. DOI: 10.1006/jvlc.2002.0232.

[32] Helen C. Purchase. „Which aesthetic has the greatest effect on human understanding?“ In: *Graph Drawing*. Hrsg. von Springer. Berlin, Heidelberg, 1997, S. 248–261. ISBN: 978-3-540-69674-2. DOI: 10.1007/3-540-63938-1_67.

[33] Helen C. Purchase, Jo-Anne Allder und David Carrington. „Graph Layout Aesthetics in UML Diagrams: User Preferences“. In: *Journal of Graph Algorithms and Applications* 6.3 (2002), S. 255–279. DOI: 10.7155/jgaa.00054.

[34] Helen C. Purchase, Jo-Anne Allder und David Carrington. „User Preference of Graph Layout Aesthetics: A UML Study“. In: *Graph drawing*. Hrsg. von Springer. Bd. 1984. Lecture Notes in Computer Science. Berlin: Springer, 2001, S. 5–18. ISBN: 978-3-540-41554-1. DOI: 10.1007/3-540-44541-2_2.

[35] Helen C. Purchase, David Carrington und Jo–Anne Allder. „Empirical Evaluation of Aesthetics-based Graph Layout“. In: *Empirical Software Engineering* 7.3 (2002), S. 233–255. ISSN: 13823256. DOI: 10.1023/A:1016344215610.

[36] Helen C. Purchase, David A. Carrington und Jo-Anne Allder. „Experimenting with Aesthetics-Based Graph Layout“. In: *Proceedings of the First International Conference on Theory and Application of Diagrams*. Hrsg. von Springer-Verlag. Diagrams '00. Berlin, Heidelberg: Springer-Verlag, 2000, S. 498–501. ISBN: 3540679154. DOI: 10.1007/3-540-44590-0_46.

[37] Helen C. Purchase, Robert F. Cohen und Murray James. „Validating graph drawing aesthetics“. In: *Graph Drawing*. Hrsg. von Springer. 1996, S. 435–446. ISBN: 978-3-540-49351-8. DOI: 10.1007/BFb0021827.

[38] Tobias Scholz und Daniel Lübke. „Improving Automatic BPMN Layouting by Experimentally Evaluating User Preferences“. In: *New knowledge in information systems and technologies*. Hrsg. von Hojjat Adeli, Luís Paulo Reis und Álvaro Rocha. Bd. 930. Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, 2019, S. 748–757. ISBN: 978-3-030-16180-4. DOI: 10.1007/978-3-030-16181-1_70.

[39] Matthias Schrepfer u. a. „The Impact of Secondary Notation on Process Model Understanding“. In: *The Practice of Enterprise Modeling*. Hrsg. von Springer. 2009, S. 161–175. ISBN: 978-3-642-05352-8. DOI: 10.1007/978-3-642-05352-8_13.

[40] Martin Siebenhaller. „Orthogonal Graph Drawing with Constraints: Algorithms and Applications“. Dissertation. Tübingen: Eberhard-Karls-Universität Tübingen, 2009. URL: http://nbn-resolving.de/urn:nbn:de:bsz:21-opus-44480 (besucht am 06. 12. 2021).

[41] The Workflow Management Coalition. *Terminology & Glossary*. 1999. URL: https://healthcareworkflow.files.wordpress.com/2008/10/wfmc-tc-1011_term_glossary_v3.pdf (besucht am 18. 03. 2022).

[42] Jasmin Türker, Michael Völske und Thomas S. Heinze. „BPMN in the Wild: A Reprise“. In: *Proceedings of the 14th Central European Workshop on Services and their Composition (ZEUS 2022)*. Hrsg. von CEUR Workshop Proceedings. 2022.

[43] Barbara Tversky, Sol Kugelmass und Atalia Winter. „Cross-cultural and developmental trends in graphic productions“. In: *Cognitive Psychology* 23.4 (1991), S. 515–557. ISSN: 00100285. DOI: 10.1016/0010-0285(91)90005-9.

[44] Verein eCH. *BPMN-Modellierungskonventionen für die öffentliche Verwaltung*. Zürich, 2020. URL: https://www.ech.ch/de/dokument/17e089ed-48fe-42d0-b886-f595f8e52311 (besucht am 15. 10. 2021).