FernUniversität in Hagen
Fakultät für Mathematik und Informatik

AIG
Artificial Intelligence Group

Eine Implementierung zur Konstruktion von linearen Ordnungserweiterungen aus Suzumura-konsistenten Relation

Bachelorarbeit

zur Erlangung des Grades eines Bachelor of Science (B.Sc.)
im Studiengang Informatik

vorgelegt von
Tim Müller

Erstgutachter:	Dr. Kai Sauerwald Artificial Intelligence Group
Betreuer:	Dr. Kai Sauerwald Artificial Intelligence Group

Erklärung

Ich erkläre, dass ich die Bachelorarbeit selbstständig und ohne unzulässige Inanspruchnahme Dritter verfasst habe. Ich habe dabei nur die angegebenen Quellen und Hilfsmittel verwendet und die aus diesen wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht. Die Versicherung selbstständiger Arbeit gilt auch für enthaltene Zeichnungen, Skizzen oder graphische Darstellungen. Die Bachelorarbeit wurde bisher in gleicher oder ähnlicher Form weder derselben noch einer anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht. Mit der Abgabe der elektronischen Fassung der endgültigen Version der Bachelorarbeit nehme ich zur Kenntnis, dass diese mit Hilfe eines Plagiatserkennungsdienstes auf enthaltene Plagiate geprüft werden kann und ausschließlich für Prüfungszwecke gespeichert wird.

Der Veröffentlichung dieser Arbeit auf der Webseite des Lehrgebiets Künstliche Intelligenz und damit dem freien Zugang zu dieser Arbeit stimme ich ausdrücklich zu.

Für diese Arbeit erstellte Software wurde quelloffen verfügbar gemacht, ein entsprechender Link zu den Quellen ist in dieser Arbeit enthalten. Gleiches gilt für angefallene Forschungsdaten.

Filderstadt, 15.08.2025 .....

(Ort, Datum) (Unterschrift)

Zusammenfassung

In Bereichen wie der Mathematik, Informatik und Wirtschaftswissenschaften spielen binäre Relationen und deren Erweiterungen zu totalen Ordnungen eine zentrale Rolle. Während die Existenz totaler Ordnungserweiterungen für Suzumura-konsistente Relationen theoretisch seit 1976 durch Suzumura bewiesen ist, mangelt es bisher an entsprechenden praktischen Konstruktionsverfahren.

Die Analyse bestehender Beweise und Verfahren zeigt deren Grenzen auf. Sowohl Suzumuras als auch Andrikopoulos' Existenzbeweis basieren auf dem nicht-konstruktiven Zornschen Lemma, während das topologische Sortieren an den erlaubten Äquivalenzen scheitert.

Als Lösung wird in dieser Arbeit ein Verfahren entwickelt, für welches zwei bekannte Algorithmen angepasst und miteinander kombiniert werden: Zunächst unterteilt eine angepasste Variante des Tarjan-Algorithmus die Eingabe-Relation in Äquivalenzklassen und überprüft gleichzeitig die Suzumura-Konsistenz. Anschließend generiert ein erweiterter Kahn-Algorithmus alle totalen Ordnungserweiterungen durch Berücksichtigung von möglichen Äquivalenzen und Betrachtung aller nicht-leeren Teilmengen der jeweils verfügbaren Elemente.

Daraufhin zeigt eine durchgeführte Laufzeit-Analyse eine lineare Laufzeit von $O(n+m)$ pro generierter totaler Ordnungserweiterung, wobei n die Anzahl der Elemente und m die Anzahl der Präferenzen darstellt.

Die Implementierung des Verfahrens erfolgt in Java unter Verwendung einer Layered Architecture mit benutzerfreundlicher GUI und der Möglichkeit zur Ein- und Ausgabe per txt-Datei. Dabei sichert eine Streaming-Ausgabe auch bei sehr großen Ergebnismengen einen konstanten Speicherbedarf.

Anschließend zeigt die Evaluation sowohl die funktionale Korrektheit als auch die Effizienz des Verfahrens und dessen Implementierung. Zum einen bestätigt die qualitative Evaluation die korrekte Erkennung von Suzumura-Inkonsistenzen und die vollständige Generierung aller möglichen totalen Ordnungserweiterungen. Zum anderen wird auch bei der Generierung von über 100 Millionen totalen Ordnungserweiterungen eine Laufzeit von weniger als 0,1 ms pro Erweiterung erreicht.

Das entwickelte Verfahren ermöglicht vielfältige Anwendungsmöglichkeiten wie zum Beispiel in der Datenbankoptimierung und der Produktionsplanung. Zusammenfassend stellt das entwickelte Konstruktionsverfahren ein funktionsfähiges Werkzeug für die Forschung und Praxis dar.

Inhaltsverzeichnis

1 Einleitung

Binäre Relationen bilden das Fundament verschiedener Bereiche der Mathematik, Informatik und Wirtschaftswissenschaften. Insbesondere Ordnungsrelationen spielen eine zentrale Rolle, da sie es ermöglichen, Elemente einer Menge in eine bestimmte Reihenfolge zu bringen und damit sowohl theoretische als auch praktische Probleme zu strukturieren. Die Bedeutung binärer Relationen zeigt sich vor allem in ihrer Anwendbarkeit in unterschiedlichen Disziplinen [2].

In vielen praktischen Anwendungen ermöglicht die Erweiterung einer partiellen Ordnung zu einer totalen Ordnung, alle Elemente einer Menge in eine eindeutige Reihenfolge zu bringen, selbst wenn in der ursprünglichen Relation nicht alle Paare von Elementen vergleichbar sind. Ein klassisches Beispiel hierfür ist die Entscheidungstheorie, in der oft nur partielle Präferenzbeziehungen zwischen Alternativen bekannt sind, aber eine vollständige Ordnung benötigt wird [17]. Ähnliche Situationen treten in der Sozialwahltheorie auf, in der individuelle Präferenzen zu einer gesellschaftlichen Ordnung zusammengefasst werden müssen, oder in der Informatik bei der Ressourcenallokation, wo Abhängigkeitsbeziehungen zwischen Aufgaben meist nur eine partielle Ordnung aufweisen [9].

Die Konstruktion einer totalen Ordnungserweiterung für Suzumura-konsistente Relationen stellt dabei eine besonders interessante Herausforderung dar. Die Suzumura-Konsistenz wurde 1976 von Suzumura definiert und ist im Vergleich zur Transitivität eine schwächere Form der Konsistenz [23]. Im Gegensatz zur Transitivität erlaubt die Suzumura-Konsistenz das Auftreten äquivalenter Elemente innerhalb von Zyklen, solange diese keine strikten Präferenzen enthalten, und macht sie damit zu einer spannenden Konsistenz-Bedingung für Anwendungsfälle, in denen klassische Transitivitätsanforderungen zu restriktiv wären.

Die Grundlage dieser Problemstellung bildet die Arbeit von Szpilrajn aus dem Jahr 1930, der mit seinem Erweiterungstheorem die Existenz einer linearen Ordnungserweiterung für jede partielle Ordnung nachgewiesen hat [24].

Forschungsstand und theoretische Einordnung

Die Forschung zu totalen Ordnungserweiterungen und Suzumura-konsistenten Relationen hat auch in den vergangenen Jahrzehnten Fortschritte gezeigt. Nach Suzumuras grundlegender Einführung des Konsistenzkonzepts erweiterte Andrikopoulos im Jahr 2009 die Anwendbarkeit der Suzumura-Konsistenz insbesondere im ökonomischen Kontext [1]. Seine Arbeit liefert nicht nur theoretische Verallgemeinerungen, sondern auch erste Ansätze für ein konstruktives Beweisverfahren.

Im weiteren trug Cato 2012 zur Vereinfachung und Erweiterung bestehender Beweise bei, indem er eine neue Erweiterung entwickelte, die die Theoreme von Szpilrajn, Arrow und Suzumura in einem einheitlichen Rahmen zusammenführt [7].

In jüngster Vergangenheit haben Bossert et al. die theoretischen Grundlagen weiter ausgebaut und dabei insbesondere die Beziehungen zwischen Suzumura-Konsistenz, Quasi-Transitivität und anderen Kohärenzeigenschaften binärer Relationen untersucht [3], [4]. Ihre Arbeiten aus den Jahren 2023 und 2024 zeigen neue Ergebnisse zur Existenz und zu Eigenschaften von Ordnungserweiterungen, die sowohl Suzumura-konsistent als auch quasi-transitiv sind, und stellen das Konzept der minimalen quasi-transitiven Erweiterung für Suzumura-konsistente Relationen vor. Parallel dazu haben Zeller und Stevens eine verallgemeinerte Version von Szpilrajns Erweiterungstheorem unter Verwendung der Terminologie von Bossert und Suzumura dargelegt [27], [5].

Algorithmische Herausforderungen und bestehende Ansätze

Während die theoretischen Grundlagen etabliert sind, bleibt die Konstruktion totaler Ordnungserweiterungen Suzumura-konsistenter Relationen eine Herausforderung. Klassische Verfahren wie das topologische Sortieren sind zwar effizient für azyklische Relationen, aber aufgrund der erlaubten Äquivalenzen in Suzumura-konsistenten Relationen nicht anwendbar.

Ein interessanter Ansatz in diesem Kontext ist der Pruesse-Rusky-Algorithmus, der für die Konstruktion aller linearen Erweiterungen partieller Ordnungen entwickelt wurde [22]. Dieser Algorithmus verwendet eine rekursive Backtracking-Strategie, um systematisch alle möglichen linearen Erweiterungen zu generieren. Allerdings ist seine direkte Anwendung auf Suzumura-konsistente Relationen nicht ohne weiteres möglich, da er für die Behandlung äquivalenter Elemente erweitert werden müsste.

Die Entwicklung eines Verfahrens zur Berechnung totaler Ordnungserweiterungen in einer Programmiersprache wie Java bietet die Möglichkeit, die theoretischen Konzepte in die Praxis umzusetzen und ein nützliches Werkzeug für weitere Forschung und Anwendung zu schaffen. Dabei ergeben sich allerdings verschiedene Herausforderungen: Zum einen muss die Effizienz des Verfahrens gewährleistet werden, da die Anzahl möglicher totaler Ordnungserweiterungen exponentiell mit der Komplexität der Eingabe-Relation wachsen kann. Zum anderen müssen geeignete Datenstrukturen entwickelt werden, die sowohl die mathematischen Konzepte angemessen repräsentieren als auch eine intuitive Benutzeroberfläche (engl. graphical user interface, kurz GUI) ermöglichen.

Zielsetzung und Beitrag der Arbeit

Diese Arbeit verfolgt das Ziel, basierend auf den theoretischen Existenzbeweisen sowie bekannten Verfahren ein praktisches Konstruktionsverfahren zu entwickeln. Konkret soll ein Verfahren entwickelt und implementiert werden, das aus einer gegebenen Suzumura-konsistenten Relation alle möglichen totalen Ordnungserweite-

rungen konstruiert. Dabei soll nicht nur eine einzelne totale Ordnungserweiterung gefunden werden, wie es viele bestehende Verfahren leisten, sondern die vollständige Menge aller totalen Ordnungserweiterungen.

Durch die Kombination bewährter Algorithmen aus der Graphentheorie - insbesondere einer angepassten Versionen des Tarjan-Algorithmus zur Erkennung stark zusammenhängender Komponenten (engl. strongly connected components, kurz SCC) und des Kahn-Algorithmus zum topologischen Sortieren - entsteht ein Verfahren, das gezielt auf die Eigenschaften Suzumura-konsistenter Relationen eingeht.

Aufbau der Arbeit

Die Arbeit gliedert sich in sieben Kapitel, die systematisch von den theoretischen Grundlagen zur praktischen Entwicklung und Implementierung führen.

Im Anschluss an diese Einleitung werden in Kapitel 2 die notwendigen theoretischen Grundlagen der binären Relationen beschrieben. Nach der Definition binärer Relationen werden zunächst die wichtigsten Eigenschaften wie Transitivität, Reflexivität, Symmetrie und Totalität eingeführt. Anschließend werden besondere Teil-/Relationen behandelt, die für das weitere Verständnis erforderlich sind. Ein zentraler Abschnitt widmet sich der Darstellung der Suzumura-Konsistenz und ihrer Abgrenzung zu anderen Konsistenzbedingungen wie der Δ-Konsistenz und der klassischen Transitivität. Das Kapitel schließt mit der Definition relevanter Erweiterungen, von der einfachen Erweiterung einer Relation bis hin zur linearen Ordnungserweiterung, und der Erläuterung, weshalb im Allgemeinen für eine Suzumura-konsistente Relation keine linearen sondern nur totale Ordnungserweiterungen konstruiert werden können.

Kapitel 3 untersucht bestehende Ansätze zur Konstruktion von Ordnungserweiterungen und bewertet ihre Eignung für Suzumura-konsistente Relationen. Nach der Einführung eines durchgängigen Beispielfalls werden zunächst Suzumuras und Andrikopoulos' Existenzbeweis analysiert. Beide Beweise werden hinsichtlich ihrer praktischen Umsetzbarkeit bewertet, wobei sich zeigt, dass sie für ein konstruktives Verfahren nicht geeignet sind. Abschließend wird das topologische Sortieren betrachtet und seine Grenzen bei der Behandlung äquivalenter Elemente aufgezeigt.

Kapitel 4 bildet den Kern der methodischen Entwicklung. Basierend auf den Erkenntnissen aus Kapitel 3 wird ein Verfahren entwickelt, das die Vorteile bestehender Beweise und Verfahren kombiniert und deren Schwächen überwindet. Die Entwicklung beginnt mit einer theoretischen Herleitung des Verfahrens, gefolgt von wichtigen Architekturentscheidungen für die spätere Implementierung. Detailliert werden die Implementierungen der Kern-Algorithmen beschrieben: der angepasste Tarjan-Algorithmus zur gleichzeitigen Äquivalenzklassenbildung und Suzumura-Konsistenz-Prüfung sowie der erweiterte Kahn-Algorithmus zur Generierung aller totalen Ordnungserweiterungen. Eine Laufzeitanalyse bewertet die Effizienz des

entwickelten Verfahrens, wobei sowohl die Laufzeit pro Ordnungserweiterung als auch die Gesamtlaufzeit betrachtet werden. Ebenso werden praktische Aspekte der Implementierung, insbesondere die GUI und Eingabevalidierung, behandelt. Das Kapitel schließt mit einer veranschaulichenden Anwendung des entwickelten Verfahrens auf den Beispielfall.

Kapitel 5 beschreibt die konkrete Umsetzung des entwickelten Verfahrens in Java unter Verwendung einer Layered Architecture. Der Domain Layer modelliert die fachlichen Konzepte wie Elemente und Präferenzen als Java-Objekte. Der Infrastructure Layer enthält die algorithmischen Kernimplementierungen, während der Application Layer die Geschäftslogik und Workflow-Koordination übernimmt. Zuletzt realisiert der Presentation Layer die GUI als auch die Eingabevalidierung.

In Kapitel 6 wird die entwickelte Implementierung sowohl qualitativ als auch quantitativ evaluiert. Die qualitative Evaluation zeigt die korrekte Funktionsweise anhand verschiedener Testfälle, einschließlich der Erkennung von Suzumura-Inkonsistenzen während die quantitative Evaluation die Performance bei großen Ergebnismengen untersucht und die theoretischen Laufzeitanalysen durch praktische Messungen validiert. Dabei werden auch die theoretischen Grenzen des Verfahrens berücksichtigt, insbesondere die durch Fubini-Zahlen und Delannoy-Zahlen beschriebenen exponentiellen Wachstumsraten der Ergebnismengen.

Kapitel 7 fasst die Ergebnisse zusammen, diskutiert die Grenzen des entwickelten Ansatzes und gibt Empfehlungen für zukünftige Entwicklungen und Anwendungen.

2 Grundlagen der binären Relationen

In dem folgenden Kapitel werden die grundlegenden Informationen zu binären Relationen und ihren Eigenschaften beschrieben, die für das Verständnis der weiteren Arbeit erforderlich sind. Dabei wurde sich an [10] orientiert. Zunächst wird in Kapitel 2.1 definiert, was eine binäre Relation ist und darauf folgend in Kapitel 2.2 beschrieben, welche Eigenschaften eine binäre Relation haben kann. In Kapitel 2.3 werden besondere Teil-/Relationen beschrieben, die in den in Kapitel 3 vorgestellten Grundlagen zur Konstruktion einer linearen Erweiterung Verwendung finden. Im Anschlus wird in Kapitel 2.4 die Suzumura-Konsistenz vorgestellt und mit weiteren Konsistenz-Bedingungen verglichen. Den Abschluss bildet Kapitel 2.5 mit der Definition einer Erweiterung einer Relation und von Erweiterungen, die gewisse Voraussetzungen erfüllen.

2.1 Definition binärer Relationen

Binäre Relationen bilden ein fundamentales Konzept mit vielfältigen Anwendungen in der Informatik, Mathematik und den Sozialwissenschaften. Sie dienen der

Modellierung von Beziehungen zwischen Objekten und finden bspw. Verwendung in der Darstellung von Präferenzen, der Modellierung von Datenbeziehungen sowie der Darstellung von Netzwerkstrukturen.

Definition 2.1. *Eine binäre Relation (im weiteren Verlauf dieser Arbeit nur noch „Relation“ genannt) R ist eine Teilmenge des kartesischen Produkts zweier nicht-leerer Mengen $X \times Y$, wobei eine Präferenz $(a, b) \in R$ als „a steht in Relation zu b“ (auch notiert als aRb) interpretiert wird.*

Handelt es sich bei der Relation R um die Teilmenge des kartesischen Produkts einer Menge X mit sich selbst, spricht man von einer homogenen Relation auf der Menge X; andernfalls von einer heterogenen Relation.

Im weiteren Verlauf dieser Arbeit wird nur die homogene Relation betrachtet. Zur Veranschaulichung einer Relation wird oft ein gerichteter Graph verwendet, wobei die Elemente als Knoten und die Präferenzen als gerichtete Kanten dargestellt werden.

2.2 Eigenschaften binärer Relationen

Relationen können gewissen Eigenschaften erfüllen, die darüber entscheiden, ob Relationen konsistent sind oder sich zu speziellen Ordnungen erweitern lassen. Die wichtigsten Eigenschaften werden im Folgenden definiert und am Beispiel eines gerichteten Graphen beschrieben.

Transitivität

Die erste Eigenschaft von Relationen ist die Transitivität.

Definition 2.2. *Eine Relation $R \subseteq X \times X$ heißt transitiv, wenn für alle Elemente $x, y, z \in X$, für die $(x, y) \in R$ und $(y, z) \in R$ gilt, auch $(x, z) \in R$ gilt.*

In einem gerichteten Graphen heißt dies, dass es für je zwei Knoten, die über einen dritten Knoten miteinander verbunden sind, auch eine direkte Kante zwischen diesen beiden Knoten existiert. Ein Beispiel für eine transitive Relation ist die Kleiner-Relation < auf den reellen Zahlen, da für drei reelle Zahlen x, y, z aus $x < y$ und $y < z$ immer $x < z$ folgt.

Intransitive Relationen erfüllen diese Implikation nicht für alle Tripel von Elementen, wie Definition 2.3 zeigt.

Definition 2.3. *Eine Relation $R \subseteq X \times X$ heißt intransitiv, wenn Elemente $x, y, z \in X$ existieren, für die $(x, y) \in R$ und $(y, z) \in R$ nicht $(x, z) \in R$ gilt.*

Ein Beispiel für eine intransitive Relation ist das Spiel „Schere-Stein-Papier“. Wäre hier Transitivität gegeben, würde aus „Schere schlägt Papier“ und „Papier schlägt

Stein“ „Schere schlägt Stein“ folgen. Allerdings gilt nur die entgegengesetzte Regel „Stein schlägt Schere“, was deutlich macht, dass die Transitivität nicht gegeben ist.

Reflexivität

Eine ebenso wichtige Eigenschaft von Relationen ist die Reflexivität.

Definition 2.4. *Eine Relation $R \subseteq X \times X$ ist reflexiv, wenn in R jedes Element aus X zu sich selbst in Relation steht.*

Ein Beispiel hierfür ist die Gleichheitsrelation =. In einem gerichteten Graphen lässt sich die Reflexivität durch Kanten von jedem Knoten zu sich selbst darstellen.

Bei irreflexiven Relationen wie bspw. der Kleiner-Relation < ist mindestens eine reflexive Präferenz (x, x) nicht enthalten.

Definition 2.5. *Eine Relation $R \subseteq X \times X$ ist irreflexiv, wenn es in X mindestens ein Element gibt, welches nicht zu sich selbst in Relation steht.*

Symmetrie

Die nächste Eigenschaft von Relationen, die betrachtet wird, ist die Symmetrie.

Definition 2.6. *Eine Relation $R \subseteq X \times X$ heißt symmetrisch, wenn für alle Elemente $x, y \in X$ aus $(x, y) \in R$ stets auch $(y, x) \in R$ folgt.*

Diese Eigenschaft lässt sich in einem gerichteten Graphen folgendermaßen veranschaulichen: Zu jeder Kante von einem Knoten x zu einem Knoten y existiert auch die entgegengesetzte Kante von y nach x.

Ein Beispiel hierfür ist die Relation „hat die gleiche Augenfarbe wie“, denn hier folgt aus „Person A hat die gleiche Augenfarbe wie Person B“ immer auch „Person B hat die gleiche Augenfarbe wie Person A“.

Im Gegensatz zu den vorherigen Eigenschaften gibt es bei der Symmetrie nicht nur zwei mögliche Fälle (symmetrisch und nicht symmetrisch), sondern drei Unterscheidungen. Neben der Symmetrie gibt es die Eigenschaften der Asymmetrie und der Antisymmetrie.

Definition 2.7. *Eine Relation $R \subseteq X \times X$ heißt asymmetrisch, wenn für alle Elemente $x, y \in X$ aus $(x, y) \in R$ stets $(y, x) \notin R$ folgt – unabhängig davon, ob $x = y$ oder $x \neq y$ gilt.*

Dies bedeutet, dass die Reflexivität bei einer asymmetrischen Relation ausgeschlossen wird. Ein Beispiel hierfür ist die Kleiner-Relation <. Dagegen gilt:

Definition 2.8. *Eine Relation $R \subseteq X \times X$ heißt antisymmetrisch, wenn für alle Elemente $x, y \in X$ mit $x \neq y$ aus $(x, y) \in R$ stets $(y, x) \notin R$ folgt.*

Im Gegensatz zur Asymmetrie ist für den Fall $x = y$ sowohl $(x, y) \in R$ als auch $(y, x) \in R$ erlaubt. Damit ist Reflexivität gestattet. Ein Beispiel für eine antisymmetrische Relation ist die Kleiner-Gleich-Relation $\leq$.

Totalität/Vollständigkeit

Bei der Eigenschaft der Totalität bzw. Vollständigkeit unterscheidet Andrikopoulos in „complete“ und „strongly complete“ [1].

Definition 2.9. *Eine Relation heißt nach Andrikopoulos vollständig ("complete"), wenn für alle Elemente $x, y \in X$ mit $x \neq y$ entweder $(x, y) \in R$ oder $(y, x) \in R$ gilt.*

Ein Beispiel hierfür ist die Kleiner-Relation < auf den reellen Zahlen. Diese ist vollständig, da für jedes Paar von reellen Zahlen eine der beiden kleiner ist als die andere, zwei unterschiedliche Zahlen aber nie gleichwertig sind.

Definition 2.10. *Eine Relation heißt nach Andrikopoulos stark vollständig („strongly complete“), wenn für alle Elemente $x, y \in X$ entweder $(x, y) \in R$ oder $(y, x) \in R$ gilt.*

Ein Beispiel hierfür ist die Kleiner-Gleich-Relation $\leq$ auf den reellen Zahlen, da sie neben der „einfachen“ Vollständigkeit durch die Kleiner-Relation < durch die Gleichheits-Relation = auch äquivalente Elemente enthalten kann und die Gleichheit für dieselbe Zahl abbildet (Reflexivität).

Der Unterschied besteht darin, dass bei der Definition der Vollständigkeit die Voraussetzung $x \neq y$ gilt, was bei der starken Vollständigkeit nicht gilt. Das heißt, dass stark vollständige Relationen auch reflexiv und symmetrisch sein können, was bei vollständigen Relationen nach Andrikopoulos' Definition verboten ist. Am Beispiel eines gerichteten Graphen bedeutet dies, dass bei einer stark vollständigen Relation sowohl zwischen jedem Paar von Knoten mindestens eine Kante existiert als auch jeder Knoten eine Kante zu sich selbst besitzt. Bei der „einfachen“ Vollständigkeit muss zwischen jedem Paar von Knoten genau eine Kante existieren und es gibt keine Kanten von einem Knoten zu sich selbst.

Zyklizität

Die letzte hier betrachtete Eigenschaft von Relationen ist die Zyklizität.

Definition 2.11. *Eine Relation ist zyklisch, wenn es eine endliche Folge von verschiedenen Elementen $x_1, x_2, \ldots, x_n \in X$ gibt, sodass jedes Element durch eine in der Relation enthaltenen Präferenz mit dem nächsten verbunden ist und das letzte Element wieder zum Ersten führt.*

In einem gerichteten Graphen würden die entsprechenden Kanten einen Kreis darstellen. Ein einfaches Beispiel einer zyklischen Relation, ist erneut das Spiel „Stein-

Schere-Papier“, da die drei Regeln „Schere schlägt Papier“, „Papier schlägt Stein“ und „Stein schlägt Schere“ einen Zyklus bilden.

Das Gegenteil bilden azyklische Relationen. Diese schließen solche Zyklen aus.

Definition 2.12. *Eine Relation ist azyklisch, wenn es keine endliche Folge von verschiedenen Elementen $x_1, x_2, \ldots, x_n \in X$ gibt, sodass jedes Element durch eine in der Relation enthaltenen Präferenz mit dem nächsten verbunden ist und das letzte Element wieder zum ersten führt.*

Ein Beispiel hierfür ist die Kleiner-Relation <, da es niemals eine Folge von Präferenzen geben kann, bei der ein Element über t Zwischenschritte wieder zu sich selbst „zurückkommt“.

2.3 Besondere Teil-/Relationen

Neben den vorgestellten Eigenschaften von Relation werden nun ausgewählte Teil-Relationen vorgestellt, die im weiteren Verlauf dieser Arbeit von Relevanz sind. Dazu werden zuerst zwei Teil-Relationen - der symmetrische und der asymmetrische Teil einer Relation - vorgestellt. Im Anschluss folgen die diagonale Relation und die transitive Hülle einer Relation.

Symmetrischer Teil einer Relation

Als Erstes wird hier der symmetrische Teil einer Relation definiert.

Definition 2.13. *Der symmetrische Teil einer Relation R wird mit $I(R)$ bezeichnet und umfasst alle Präferenzen $(x, y) \in R$, für die auch $(y, x) \in R$ gilt.*
Ist eine Präferenz (x, y) Teil von $I(R)$, schreibt man auch xIy.

Dies entspricht dem Schnitt $R \cap R^{-1}$, wobei R^{-1} die Umkehrrelation von R ist. Komplett symmetrische Relationen lassen sich durch $R = R^{-1}$ charakterisieren.

Asymmetrischer Teil einer Relation

Das Gegenteil des symmetrischen Teils einer Relation ist der asymmetrische Teil.

Definition 2.14. *Der asymmetrische Teil einer Relation R wird mit $P(R)$ bezeichnet und umfasst alle Präferenzen $(x, y) \in R$, für die $(y, x) \in R$ nicht gilt.*
Ist eine Präferenz (x, y) Teil von $P(R)$, schreibt man auch xPy.

Asymmetrische Relationen erfüllen die Bedingung $R \cap R^{-1} = \emptyset$ und sind stets irreflexiv, da sie auch die diagonalen Präferenzen wie (x, x) nicht enthalten.

Jede Relation lässt sich in einen symmetrischen und einen asymmetrischen Teil aufteilen, wobei der Schnitt beider Teile die leere Menge ist und die Vereinigung beider Teile die Ausgangsrelation darstellt.

Die Kleiner-Gleich-Relation $\leq$ lässt sich bspw. in einen symmetrischen und einen asymmetrischen Teil aufteilen. Dabei enthält der symmetrische Teil alle Präferenzen, die die Gleichheits-Bedingung erfüllen und der asymmetrische Teil enthält alle Präferenzen, die die Kleiner-Bedingung erfüllen.

Diagonale Relation

Eine Relation, die genau alle reflexiven Paare bzw. die Kanten von allen Knoten zu sich selbst beinhaltet, ist die diagonale Relation, welche oft auch nur Diagonale genannt wird.

Definition 2.15. *Die diagonale Relation auf einer Menge X wird als Δ bezeichnet und enthält genau die (reflexiven) Präferenzen (x, x) für alle $x \in X$.*

Sie ist die kleinste reflexive Relation auf X und es gilt, dass eine Relation R genau dann reflexiv ist, wenn sie die diagonale Relation Δ beinhaltet.

Transitive Hülle

Die letzte relevante besondere Relation ist die transitive Hülle.

Definition 2.16. *Die transitive Hülle einer Relation R wird als $\overline{R}$ bezeichnet und ist die kleinste transitive Relation, die R vollständig enthält. Sie ergänzt R um alle indirekten Präferenzen, die durch die Transitivitäts-Eigenschaft gebildet werden können.*

Sie wird gebildet, indem zu R alle Präferenzen (x, z) hinzugefügt werden, für die eine Kette von Präferenzen der Art $(x, y_1), (y_1, y_2), \ldots, (y_n, z) \in R$ existiert. Ebenso kann man die transitive Hülle iterativ erstellen, indem man in jeder Iteration die Präferenzen (x, z) hinzufügt, für die (x, y) und (y, z) bereits in der Relation enthalten sind. Dies führt man so lange fort, bis keine neuen Präferenzen mehr ergänzt werden.

Vereinigt man die transitive Hülle mit der zuvor vorgestellten diagonalen Relation, erhält man die reflexiv-transitive Hülle einer Relation R. Diese ist, analog zur transitiven Hülle, die kleinste reflexive und transitive Relation, die R enthält.

2.4 Suzumura-Konsistenz

Nachdem die grundlegenden Eigenschaften von Relationen und die für den weiteren Verlauf der Arbeit relevanten besonderen Teil-/Relationen beschrieben wurden, wird im Folgenden einer der wichtigsten Aspekte dieser Arbeit behandelt - die Suzumura-Konsistenz. Dazu wird in diesem Kapitel nicht nur beschrieben, wann eine Relation Suzumura-konsistent ist, sondern auch die Eigenschaft der Suzumura-Konsistenz mit zwei weiteren Konsistenz-Eigenschaften - der Δ-Konsistenz und der

bereits definierten Transitivität - verglichen, um darzustellen, weshalb die Suzumura-Konsistenz eine spannende Konsistenz-Eigenschaft im Rahmen der Konstruktion linearer Erweiterungen darstellt.

Definition der Suzumura-Konsistenz

Die Suzumura-Konsistenz ist ein bedeutendes Konzept der Sozialwahltheorie, das von Kotaro Suzumura [23] im Jahr 1976 eingeführt wurde. Sie stellt eine abgeschwächte Form der Transitivität für Relationen dar und ermöglicht dadurch eine flexiblere Modellierung individueller und kollektiver Präferenzen, insbesondere in Situationen, in denen bspw. die Transitivität zu restriktiv wäre. Suzumura definiert seine Art der Konsistenz folgendermaßen:

Definition 2.17. *Eine Relation ist Suzumura-konsistent, sofern sie keine PR-Zyklen enthält.*

Um verstehen zu können, wann eine Relation Suzumura-konsistent ist, muss geklärt werden, was ein PR-Zyklus ist. Dies wird in der folgenden Definition erklärt.

Definition 2.18. *Ein PR-Zyklus ist ein Zyklus innerhalb einer Relation, der bei einem Element x startet und nach t Schritten wieder bei x ankommt, mindestens eine strikte Präferenz enthält und folgende Struktur hat:*

$$x_1 P x_2 R \dots R x_t R x_1$$

Äquivalenzen zwischen zwei unterschiedlichen Elementen sind dementsprechend erlaubt und gefährden die Suzumura-Konsistenz nicht.

Vergleich mit der Δ-Konsistenz

Neben der Suzumura-Konsistenz spricht Andrikopoulos auch von der Δ-Konsistenz [1].

Definition 2.19. *Eine Relation R auf einer Menge X ist Δ-konsistent, wenn sie Suzumura-konsistent ist und zusätzlich die Bedingung $I(\overline{R}) \subseteq \Delta$ erfüllt, wobei $\overline{R}$ die transitive Hülle von R und Δ die diagonale Relation darstellt.*

Das bedeutet, dass die Δ-Konsistenz neben den PR-Zyklen auch äquivalente Elemente ausschließt.

Sichtbar wird der Unterschied der beiden Konsistenz-Bedingungen an folgendem Beispiel:

$$X = \{a, b, c, d, e\}$$
$$R = \{(a, b), (b, c), (c, d), (d, c)\},$$

Diese Relation enthält die äquivalenten Elemente c und d, die mit $cRdRc$ bzw. $dRcRd$ einen Zyklus bilden. Da beiden Elemente c und d äquivalent sind, handelt es sich hierbei jedoch um keinen PR-Zyklus. Damit ist diese Relation Suzumura-konsistent.

Die Δ-Konsistenz ist hingegen nicht erfüllt, da die Bedingung $I(R) \subseteq \Delta$ durch $I(R) = \{(c,d),(d,c)\} \nsubseteq \Delta$ nicht erfüllt ist.

Da die Suzumura-Konsistenz eine der beiden Bedingungen der Δ-Konsistenz darstellt, kann man festhalten, dass jede Δ-konsistente Relation auch Suzumura-konsistent ist und die Suzumura-Konsistenz somit eine schwächere Konsistenz-Bedingung darstellt. Daher stellt sie als Bedingung für die Konstruktion einer linearen Ordnungserweiterung die spannendere Konsistenz-Bedingung dar und wird im weiteren Verlauf dieser Arbeit entsprechend betrachtet.

Vergleich mit Transitivität

Neben der Δ-Konsistenz stellt auch die bereit in Kapitel 2.2 definierte Transitivität eine spannende Art der Konsistenz-Bedingung dar. Sie ist im Vergleich zur Suzumura-Konsistenz eine stärkere Eigenschaft, da sie alle Arten von Zyklen verhindert, sofern keine Äquivalenz zwischen allen Elementen des Zyklus gegeben ist. Dies wird auch intransitive Indifferenz genannt. Folgendes Beispiel verdeutlicht dies:

$$X = \{a,b,c,d\}$$
$$R = \{(a,b),(b,a),(b,c),(c,b),(c,d),(d,c),(d,a),(a,d)\}$$

Diese Relation ist Suzumura-konsistent, da sie nur aus symmetrischen Präferenzen besteht ($I(R) = R$). Allerdings ist die Relation nicht transitiv, da eine intransitive Indifferenz bzw. ein Zyklus ohne vollständige Äquivalenz zwischen den Elementen des Zyklus besteht.

Für das Erfüllen der Transitivität fehlen in der Relation R die Präferenzen (b,d) und (d,b), da aktuell $(b,c) \in R$ und $(c,d) \in R$ gilt, aber $(b,d) \notin R$. Gleiches gilt für die umgekehrte Reihenfolge der Elemente. So lässt sich festhalten, dass eine Relation Suzumura-konsistent, aber nicht transitiv sein kann.

Für die entgegengesetzte Richtung gilt dies jedoch nicht, wie an einem einfachen Beispiel gezeigt werden kann. Gegeben sei eine transitive, aber nicht Suzumura-konsistente Relation R auf einer Menge X. Dann existiert in der Relation ein PR-Zyklus der Art $x_1 P x_2 R \dots R x_t R x_1$ - also eine strikte Präferenz von x_1 zu x_2. Dementsprechend gilt aber auch $x_2 R \dots R x_t R x_1$, wodurch aus der vorausgesetzten Transitivität $x_2 R x_1$ folgt, was der angenommenen strikten Präferenz von x_1 zu x_2 widerspricht. Damit ist gezeigt, dass in einer transitiven Relation kein PR-Zyklus existieren kann und jede transitive Relation auch Suzumura-konsistent ist.

So gilt auch hier, dass die Suzumura-Konsistenz gegenüber der Transitivität die schwächere Konsistenz-Bedingung darstellt und für die folgende Betrachtung herangezogen wird.

2.5 Definition relevanter Erweiterungen

Zum Abschluss der Grundlagen zu Relationen wird in diesem Kapitel dargestellt, was man unter einer Erweiterung einer Relation versteht. Zudem werden drei Erweiterungen, die besondere Eigenschaften haben und für den weiteren Verlauf dieser Arbeit relevant sind, vorgestellt und voneinander abgegrenzt.

Erweiterung

Eine Erweiterung ist ein grundlegendes Konzept in der Relationen-Theorie und wird folgendermaßen definiert:

Definition 2.20. *Gegeben sei eine Relation R auf einer Menge X. Eine Relation R' auf X heißt Erweiterung von R, wenn sowohl sie sowohl R enthält als auch alle strikten Präferenzen von R beibehält.*

Das bedeutet, dass alle Präferenzen, die in R enthalten sind, auch in R' enthalten sind, wobei R' zusätzlich weitere Präferenzen enthalten kann. Dabei muss allerdings die in R enthaltene Ordnung der Elemente gewahrt bleiben. Erweiterungen werden genutzt, um Relationen zu vervollständigen oder zusätzliche Beziehungen zwischen Elementen herzustellen, ohne bestehende Beziehungen zu verletzen. Sie spielen eine zentrale Rolle, wenn es darum geht, partielle Relationen zu Ordnungsrelationen zu erweitern, die für viele mathematische und ökonomische Anwendungen benötigt werden.

Für ein einfaches Beispiel sei $X = \{a, b, c\}$ und $R = \{(a, b)\}$. Dann ist $R' = \{(a, b), (b, c)\}$ eine Erweiterung von R, da $R \subseteq R'$ und $P(R) \subseteq P(R')$ gilt.

Erweiterung zur Quasiordnung

Neben der einfachen Erweiterung einer Relation werden nun auch drei Formen der Erweiterung dargestellt, die weitere Eigenschaften besitzen. So kann eine Relation bspw. zu einer sog. Quasiordnung erweitert werden.

Definition 2.21. *Eine Quasiordnung, auch Präordnung genannt, ist eine Relation, die sowohl reflexiv als auch transitiv ist.*

Damit ist die Quasiordnung eine Relation, die eine gewisse Ordnungsstruktur in den Elementen der Menge X besitzt, jedoch nicht zwischen all ihren Elementen, da eine Quasiordnung nicht total sein muss. Dementsprechend kann eine Erweiterung der Relation R zu einer Quasiordnung folgendermaßen definiert werden.

Definition 2.22. *Gegeben sei eine Relation R auf einer Menge X. Eine Relation R' auf X heißt Erweiterung von R zu einer Quasiordnung, wenn zum einen sowohl $R \subseteq R'$ als auch $P(R) \subseteq P(R')$ und gilt und zum anderen R' reflexiv und transitiv ist.*

Für ein einfaches Beispiel sei $X = \{a, b, c\}$ und $R = \{(a, b)\}$. Die Relation $R' = \{(a, b), (a, a), (b, b), (c, c)\}$ ist eine Erweiterung von R zu einer Quasiordnung, da sie zum einen eine Erweiterung von R - sowohl $R \subseteq R'$ als auch $P(R) \subseteq P(R')$ sind erfüllt - und zum anderen reflexiv und transitiv ist.

Totale Ordnungserweiterung

Die Erweiterung, die im Vergleich zur Quasiordnung noch eine weitere Eigenschaft erfüllt, ist die totale Ordnungserweiterung, die auch nur Ordnungserweiterung genannt wird. Zur besseren Abgrenzung zur noch folgenden linearen Ordnungserweiterung wird im Verlauf dieser Arbeit der Begriff der totalen Ordnungserweiterung beibehalten.

Eine totale Ordnungserweiterung ist eine Erweiterung, bei der die erweiterte Relation R' Vergleich zur Quasiordnung zusätzlich total ist. Das bedeutet, dass für jedes Paar verschiedener Elemente $x, y \in X$ mindestens eine der Präferenzen (x, y) oder (y, x) in der Relation enthalten ist.

Definition 2.23. *Gegeben sei eine Relation R auf einer Menge X. Eine Relation R' auf X heißt totale Ordnungserweiterung von R, wenn zum einen sowohl $R \subseteq R'$ als auch $P(R) \subseteq P(R')$ und gilt und zum anderen R' reflexiv, transitiv und total ist.*

Bzgl. der Symmetrie-Eigenschaft der Erweiterung gibt es hier noch keine Einschränkung, wodurch Äquivalenzen zwischen verschiedenen Elementen, wie sie auch in Suzumura-konsistenten Relationen auftreten können, erlaubt sind.

Für ein einfaches Beispiel sei $X = \{a, b, c\}$ und $R = \{(a, b)\}$. Die Relation $R' = \{(a, b), (a, a), (b, b), (c, c), (a, c), (b, c), (b, a)\}$ ist eine totale Ordnungserweiterung von R, da sie zum einen eine Erweiterung von R - sowohl $R \subseteq R'$ als auch $P(R) \subseteq P(R')$ sind erfüllt - und zum anderen reflexiv, transitiv und total ist. Äquivalenzen, wie in diesem Fall die äquivalenten Elemente a und b, sind erlaubt.

Lineare Ordnungserweiterung

Die Erweiterung mit den strengsten Eigenschaften ist die lineare Ordnungserweiterung oder auch nur lineare Erweiterung genannt. Eine lineare Ordnungserweiterung ist eine totale Ordnungserweiterung, die zusätzlich antisymmetrisch ist. Äquivalenzen zwischen zwei unterschiedlichen Elementen sind also nicht mehr erlaubt. So gilt für alle $x, y \in X$, wenn $(x, y) \in R'$ und $(y, x) \in R'$, dann folgt $x = y$.

Definition 2.24. *Gegeben sei eine Relation R auf einer Menge X. Eine Relation R' auf X heißt lineare Ordnungserweiterung von R, wenn zum einen sowohl $R \subseteq R'$ als auch*

$P(R) \subseteq P(R')$ gilt und zum anderen R' reflexiv, transitiv, total und antisymmetrisch ist.

Für ein einfaches Beispiel sei $X = \{a, b, c\}$ und $R = \{(a, b)\}$. Die Relation $R' = \{(a, b), (a, a), (b, b), (c, c), (a, c), (b, c)\}$ ist eine lineare Ordnungserweiterung von R, da sie zum einen eine Erweiterung von R - sowohl $R \subseteq R'$ als auch $P(R) \subseteq P(R')$ sind erfüllt - und zum anderen reflexiv, transitiv, total und antisymmetrisch ist. Äquivalenzen, wie im Beispiel der totalen Ordnungserweiterung, sind hier nicht erlaubt.

Somit ist an dieser Stelle eine wichtige Erkenntnis festzuhalten. Zum einen kann eine Suzumura-konsistente Relation äquivalente Elemente enthalten, sofern kein PR-Zyklus in der Relation enthalten ist, und zum anderen dürfen für die Konstruktion einer Erweiterung nur weitere Präferenzen zu einer Relation hinzugefügt und explizit keine Präferenzen aus einer Relation entfernt werden. Eine lineare Ordnungserweiterung, deren Konstruktion im Rahmen dieser Arbeit untersucht werden soll, fordert jedoch die Eigenschaft der Antisymmetrie, wodurch äquivalente Elemente verboten sind.

Dies hat zur Folge, dass für eine Suzumura-konsistente Relation im Allgemeinen keine lineare, sondern nur eine totale Ordnungserweiterung konstruiert werden kann. Wäre die Relation Suzumura-konsistent und würde keine äquivalenten Elemente beinhalten, wäre sie bereits Δ-konsistent. Infolgedessen wird im weiteren Verlauf dieser Arbeit die Konstruktion von totalen Ordnungserweiterungen zu Suzumura-konsistenten Relationen untersucht.

3 Grundlagen zur Konstruktion einer totalen Ordnungserweiterung

Im Rahmen dieses Kapitels werden die Grundlagen zur Konstruktion einer totalen Ordnungserweiterung zu einer Suzumura-konsistenten Relation betrachtet. Hierfür wird zu Beginn in Kapitel 3.1 ein geeigneter Beispielfall vorgestellt, mithilfe dessen im weiteren Verlauf die betrachteten Beweise und Methoden veranschaulicht werden. In Kapitel 3.2 wird der Beweis von Suzumura für die Existenz einer totalen Ordnungserweiterung einer Suzumura-konsistenten Relation beschrieben und dahingehend bewertet, inwiefern er sich als konstruktives Verfahren eignet. Dieselbe Betrachtung von Andrikopoulos' entsprechendem Beweis findet in Kapitel 3.3 statt. Den Abschluss des Kapitels bildet die Betrachtung des topologischen Sortierens zur Konstruktion einer linearen Ordnungserweiterung zu einer Suzumura-konsistenten Relation in Kapitel 3.4.

3.1 Beispielfall

Als Beispielfall wird folgende Menge X und Relation R angenommen:

$$X = \{a, b, c, d, e\}$$
$$R = \{(a, b), (b, c), (c, d), (d, c)\}$$

Dieses Beispiel ist sehr einfach und überschaubar und bietet sehr gute Eigenschaften, um die Möglichkeit der Konstruktion von totalen Ordnungserweiterungen anhand der betrachteten Beweise und Methoden zu untersuchen.

Zum einen ist die Relation R Suzumura-konsistent. Um dies zu beweisen, muss gezeigt werden, dass die Relation keine PR-Zyklen enthält. Wie schnell zu sehen ist, ist $c \rightarrow d \rightarrow c$ bzw. $d \rightarrow c \rightarrow d$ der einzige Zyklus in R. Durch $(c, d) \in R$ und $(d, c) \in R$ gilt $(c, d) \in I(R)$ und $(d, c) \in I(R)$. Das bedeutet, dass die beiden Elemente c und d äquivalent sind und damit in R kein PR-Zyklus existiert und die Relation R Suzumura-konsistent ist.

Dem entgegen ist die Relation R nicht Δ-konsistent. Für das Erreichen der Δ-Konsistenz ist neben der Suzumura-Konsistenz erforderlich, dass es keine äquivalenten Elemente gibt. Wie bereits für den Beweis der Suzumura-Konsistenz gezeigt wurde, enthält R aber die äquivalenten Elemente c und d - $(c, d), (d, c) \in R$. Somit ist R Suzumura- aber nicht Δ-konsistent.

Die letzte Eigenschaft, die sich sehr gut für die Veranschaulichung anbietet, ist, dass R ein Element besitzt, welches zu keinem der anderen Elemente vergleichbar ist. Ist dies in der Ausgangsrelation der Fall, kann es aufgrund der bei der Suzumura-Konsistenz und der totalen Ordnungserweiterung erlaubten Äquivalenzen zwischen zwei Elementen schnell zu einer großen Zahl erlaubter totaler Ordnungserweiterungen kommen.

3.2 Suzumura

Als Erstes wird der ursprüngliche Beweis von Suzumuras Theorem 3 aus seiner Arbeit von 1976, welches hier in Theorem 3.1 wiedergegeben ist, betrachtet [23].

Theorem 3.1. *Eine binäre Relation R hat eine totale Ordnungserweiterung R^* genau dann, wenn sie Suzumura-konsistent ist.*

Der Beweis dieses Theorems ist grundlegend für die Theorie der Ordnungserweiterungen, da er zeigt, dass die Suzumura-Konsistenz eine notwendige und hinreichende Bedingung für die Existenz einer totalen Ordnungserweiterung ist. Im Folgenden wird der Beweis Schritt für Schritt beschrieben und daraufhin untersucht, inwiefern er sich zur Konstruktion von totalen Ordnungserweiterungen aus Suzumura-konsistenten Relationen eignet. Dabei wird nur der Teil des Beweises untersucht, der zeigt, dass eine Suzumura-konsistente Relation eine totale Ordnungs-

erweiterung hat. Die gegensätzliche Richtung des Beweises ist für diese Arbeit nicht von Relevanz.

Um die Existenz einer totalen Ordnungserweiterung für jede Suzmura-konsistente Relation zu zeigen, betrachtet Suzumura in seinem Beweis die folgende Ausgangssituation: Gegeben sei eine nicht notwendigerweise vollständige und nicht notwendigerweise transitive binäre Relation R auf einer Menge X, von der lediglich angenommen wird, dass sie Suzumura-konsistent ist. Das Ziel ist es, zu zeigen, dass es eine totale, reflexive und transitive Relation R^* auf X gibt, die R enthält, also eine totale Ordnungserweiterung von R existiert.

Zunächst wird die transitive Hülle $\overline{R}$ von R gebildet, wodurch Transitivität sichergestellt wird. Um zusätzlich die Reflexivität sicherzustellen, wird die Diagonalrelation Δ hinzugefügt. Die so entstehende Relation

$$Q = \overline{R} \cup \Delta$$

ist somit reflexiv, transitiv und enthält R, wodurch sie eine Erweiterung von R zu einer Quasiordnung darstellt. Um zu beweisen, dass Q tatsächlich eine Quasiordnung ist, werden die Eigenschaften Reflexivität und Transitivität im Folgenden bewiesen. Reflexivität ist offensichtlich, da die Diagonalrelation einen Teil der Vereinigung bildet, mithilfe welcher Q konstruiert wurde. Für den Beweis der Transitivität wird angenommen, dass $(x, y), (y, z) \in Q$.

Im Falle, dass $(x, y), (y, z) \in \overline{R}$, gilt auch $(x, z) \in \overline{R} \subseteq Q$, da in $\overline{R}$ alle transitiven Präferenzen enthalten sind.

Falls $(x, y) \in \Delta$, gilt $x = y$, sodass $(x, z) \in Q$ aus $(y, z) \in Q$ folgt. Gleiches gilt für den Fall, dass $(y, z) \in \Delta$. In diesem Fall gilt $y = z$, sodass $(x, y) \in Q$ aus $(x, z) \in Q$ folgt. Somit ist bewiesen, dass Q sowohl reflexiv als auch transitiv und somit eine Quasiordnung ist.

Im nächsten Schritt verweist Suzumura explizit auf ein zentrales Resultat der Präferenztheorie, nämlich auf Fishburns Lemma 15.4 [11]. Dieses ist ein indirekter Verweis auf das klassische Szpilrajn-Theorem (1930), das die Existenz einer totalen Ordnungserweiterung für jede Quasiordnung garantiert [24]. Suzumuras Argumentation baut somit auf diesem grundlegenden Resultat auf und verallgemeinert es um das Kriterium der Suzumura-Konsistenz.

Lemma 3.2. *Ist die Relation Q eine Quasiordnung, dann hat sie eine Erweiterung Q', die eine totale Ordnung ist.*

Der Beweis dieses Lemmas wird Hansson in [15] geliefert und in Folgenden dargestellt.

Basierend auf der Relation Q wird zunächst die Menge W aller partiellen Relationen (reflexiv und transitiv) K gebildet, für die die folgenden beiden Voraussetzungen

erfüllt sind:

$$xQy \Rightarrow xKy$$
$$xQy \wedge \neg yQx \Rightarrow xKy \wedge \neg yKx$$

Diese Menge W ist zum einen eine nicht-leere Menge, da sie mindestens die Relation Q enthält, und zum anderen sind die in W enthaltenen Relationen durch Mengeninklusion sortiert. Das bedeutet, dass jede Relation $K \in W$ eine Teilmenge ihrer Folge-Relation ist.

Im Weiteren zeigt Hansson, dass diese Menge eine obere Schranke besitzt. Durch den Nachweis dieser Eigenschaft kann das Zornsche Lemma angewendet werden, welches besagt, dass die Menge W ein maximales Element Q' besitzt. Für diese Relation Q' beweist Hansson mithilfe eines Widerspruchsbeweises die Totalität, wodurch Q' eine totale Ordnungserweiterung zu R darstellt und Theorem 3.1 bewiesen ist.

Da die Beweisschritte ab der Konstruktion der Menge W für eine Schritt-für-Schritt-Konstruktion nicht geeignet sind, wurden diese hier nicht im Detail beschrieben. Für eine detailliertere Beschreibung dieser Beweisschritte wird auf die Darstellung von Andrikopoulos' Beweis in Kapitel 3.3 verwiesen.

Damit ist sichergestellt, dass es eine totale Ordnung Q' gibt, die Q und damit auch R enthält.

Um letztendlich noch zu beweisen, dass diese totale Ordnungserweiterung von Q auch eine totale Ordnungserweiterung von R ist, schließt Suzumura mit dem Beweis, dass Q eine Erweiterung von R ist und die Suzumura-Konsistenz von R auch in Q gewahrt bleibt. Durch die Konstruktion von Q als Vereinigung der transitiven Hülle $\overline{R}$ von R mit der Diagonalrelation ist offensichtlich, dass $R \subseteq Q$ gilt. Dass Q weiterhin Suzumura-konsistent ist, beweist Suzumura damit, dass der asymmetrische Teil von R $P(R)$ Teil des asymmetrischen Teils von Q $P(Q)$ ist, wodurch die strikten Präferenzen aus R in Q bewahrt bleiben und damit auch die Suzumura-Konsistenz von R für Q gilt.

Für den Beweis von $P(R) \subseteq P(Q)$ wird $(x, y) \in P(R)$ angenommen. Daraus folgt $(x, y) \in R$ und $(y, x) \notin R$. Aus $(x, y) \in R$ folgt $(x, y) \in Q$, wodurch nur noch $(y, x) \notin Q$ zu beweisen ist. Dies wird nun mittels Widerspruchsbeweis gezeigt. So wird zunächst angenommen, dass $(y, x) \in Q$. Offensichtlich gilt $(y, x) \notin \Delta$, da in diesem Fall $(x, y) \in P$ nicht gelten könnte. Daher bleibt nur die Möglichkeit, dass $(y, x) \in \overline{R}$. Betrachtet man zusätzlich die anfängliche Annahme $(x, y) \in P(R)$, würde dies einen PR-Zyklus ergeben, was der Suzumura-Konsistenz von R bzw. Q widerspricht. Somit ist bewiesen, dass $P(R) \subseteq P(Q)$ gilt und über die konstruierte Quasiordnung Q und Lemma 3.2 eine totale Ordnungserweiterung von R existiert.

Obwohl Suzumura in Verbindung mit Lemma 3.2 einen eleganten und allgemeinen Existenzbeweis für totale Ordnungserweiterungen Suzumura-konsistenter Relatio-

nen liefert, hat diese Methode für eine praktische, schrittweise Konstruktion erhebliche Nachteile.

Der zentrale Nachteil liegt im nicht-konstruktiven Charakter des Beweises: Hansson bildet die Menge aller partiellen Relationen, die die Ausgangsrelation enthalten, und zeigt mithilfe des Zornschen Lemmas lediglich die Existenz eines maximalen Elements. Diese Herangehensweise ist für eine algorithmische Umsetzung ungeeignet, da sie keine konkrete Berechnungsvorschrift liefert.

Ein weiteres Problem ist die exponentielle Komplexität: Die Menge aller partiellen Relationen, die die Ausgangsrelation erweitern, wächst im Worst Case mit der Größe der Grundmenge exponentiell [6]. Für praktische Zwecke ist es daher unmöglich, diese Menge zu konstruieren oder zu durchsuchen.

Schließlich sind die Beweisschritte wie Mengenbildung und Maximalitätsargument rein mengentheoretisch und lassen sich nicht direkt in ein effiziente Verfahren übersetzen. Insbesondere fehlt eine Vorschrift, wie fehlende Präferenzen schrittweise ergänzt werden können, ohne die Suzumura-Konsistenz zu verletzen.

Zusammenfassend ist der Existenzbeweis von Suzumura zwar von großer theoretischer Bedeutung, für die konkrete algorithmische Konstruktion von Ordnungserweiterungen aber nicht geeignet.

Beispiel

Nun wird der Beweis von Suzumura am in Kapitel 3.1 vorgestellten Beispiel Schritt für Schritt angewendet und veranschaulicht. Die betrachtete Menge X und Relation R sind:

$$X = \{a, b, c, d, e\}$$
$$R = \{(a, b), (b, c), (c, d), (d, c)\}$$

Im ersten Schritt wird die Vereinigung der transitiven Hülle $\overline{R}$ und der diagonalen Relation Δ gebildet. Hierfür wird zuerst die Konstruktion der transitiven Hülle und dann die Vereinigung mit der diagonalen Relation beschrieben.

Zur Konstruktion der transitiven Hülle wird in jeder Iteration untersucht, ob es Elemente $x, y, z \in X$ gibt, für die $(x, y), (y, z) \in R$ gilt, aber $(x, z) \notin R$. Ist dies der Fall, wird (x, z) der transitiven Hülle hinzugefügt.

In der ersten Iteration werden folgende Präferenzen der transitiven Hülle hinzuge-

fügt:

$$(a,b) \in R \wedge (b,c) \in R \Rightarrow (a,c) \in \overline{R}$$
$$(b,c) \in R \wedge (c,d) \in R \Rightarrow (b,d) \in \overline{R}$$
$$(c,d) \in R \wedge (d,c) \in R \Rightarrow (c,c) \in \overline{R}$$
$$(d,c) \in R \wedge (c,d) \in R \Rightarrow (d,d) \in \overline{R}$$

In der zweiten Iteration wird dann noch folgendes Paar gefunden:

$$(a,c) \in \overline{R} \wedge (c,d) \in R \Rightarrow (a,d) \in \overline{R}$$

Im weiteren Durchlauf wird keine hinzuzufügende Präferenz mehr gefunden. Daraus folgt die transitive Hülle:

$$\overline{R} = \{(a,b),(b,c),(c,d),(d,c),(a,c),(b,d),(c,c),(d,d),(a,d)\}$$

Im nächsten Schritt wird die diagonale Relation Δ gebildet, die alle reflexiven Präferenzen auf Basis der Menge X enthält:

$$\Delta = \{(a,a),(b,b),(c,c),(d,d),(e,e)\}$$

Die Vereinigung $Q = \overline{R} \cup \Delta$ ist damit automatisch eine Quasiordnung - transitiv und reflexiv - und ergibt:

$$\begin{aligned} Q &= \overline{R} \cup \Delta \\ &= \{(a,b),(b,c),(c,d),(d,c),(a,c),(b,d),(c,c),(d,d),(a,d)\} \\ &\cup \{(a,a),(b,b),(c,c),(d,d),(e,e)\} \\ &= \{(a,b),(b,c),(c,d),(d,c),(a,c),(b,d),(c,c),(d,d),(a,d),(a,a),(b,b),(e,e)\} \end{aligned}$$

Jetzt wird im Beweis von Lemma 3.2 die Menge aller Suzumura-konsistenten Erweiterungen von Q gebildet, die durch Mengeninklusion sortiert sind. Durch Anwendung des Zornschen Lemmas folgt dann, dass diese Menge mindestens ein Maximum haben muss. Dieses Maximum ist die totale Ordnungserweiterung R^* zur Eingangs-Relation R. Der von Suzumura bzw. Hansson beschriebene Beweis zeigt jedoch nur die Existenz und gibt keine Konstruktionsvorschrift an. Eine mögliche totale Ordnungserweiterung ist:

$$\begin{aligned} R^* &= Q \cup \{(a,e),(b,e),(c,e),(d,e)\} \\ &= \{(a,b),(b,c),(c,d),(d,c),(a,c),(b,d),(c,c),(d,d),(a,d),(a,a),(b,b),(e,e),(a,e), \\ &(b,e),(c,e),(d,e)\} \end{aligned}$$

Nun sind alle Elemente mit e vergleichbar, und die Relation ist total, jedoch nicht antisymmetrisch, da vorhandene Äquivalenzen erhalten bleiben - in diesem Beispiel c und d, da $(c,d),(d,c) \in R^*$.

Die angegebene Erweiterung ist damit eine totale Ordnungserweiterung von R, jedoch nur eine von sieben möglichen, da man e nicht nur am Ende der Ordnung platzieren kann, sondern auch vor, zwischen und äquivalent zu den anderen Elementen.

3.3 Andrikopoulos

Andrikopoulos behandelt die Existenz totaler Ordnungserweiterungen von Suzumura-konsistenten Relation in seiner Arbeit in Corollary 5, welches in Korollar 3.3 dargestellt ist, ebenso [1].

Korollar 3.3. *Sei R eine binäre Relation auf X und seien x, y zwei beliebige, nicht vergleichbare Elemente bezüglich R. Dann besitzt R eine Ordnungserweiterung R^*, in der xR^*y gilt, ..., genau dann, wenn R konsistent ist.*

Das bedeutet, dass für jede konsistente Relation R auf einer Menge X eine totale Ordnungserweiterung existiert.

Der Beweis von Andrikopoulos kombiniert dabei mehrere mathematische Techniken und algorithmische Ideen. Zunächst wird die gegebene Relation R in Äquivalenzklassen unterteilt, wobei zwei Elemente genau dann zur selben Äquivalenzklasse gehören, wenn sie in der reflexiv-transitiven Hülle von R äquivalent zueinander sind. Formal wird für jedes Element $s \in X$ die Äquivalenzklasse definiert als

$$[x] = \{y \in X | (x, y) \in I(\overline{R} \cup \Delta)\}$$

wobei $I(\overline{R} \cup \Delta)$ der symmetrische Teil der reflexiv-transitiven Hülle von R darstellt und Δ die Diagonale ist. Die Menge der Äquivalenzklassen $\overline{X} = \{[x] | x \in X\}$ bildet eine Partition der Grundmenge X und reduziert die Komplexität der ursprünglichen Relation.

Auf dieser neuen Menge von Äquivalenzklassen wird eine reduzierte Relation R' definiert, indem man für zwei Klassen genau dann eine Präferenzbeziehung annimmt, wenn es zwischen Repräsentanten dieser Klassen eine strikte Präferenz in der ursprünglichen Relation R gibt.

$$([x], [y]) \in R' \Leftrightarrow \exists x' \in [x], y' \in [y] : (x', y') \in P(R)$$

Diese Konstruktion ist algorithmisch darstellbar, da sie lediglich die Berechnung der transitiven Hülle und die Identifikation von Äquivalenzklassen erfordert. Die so definierte Relation R' ist antisymmetrisch und erbt die Δ-Konsistenz von R. Um die Δ-Konsistenz von R' nachzuweisen, wird angenommen, R' enthalte einen verbotenen PR-Zyklus. Dies würde bedeuten, dass es eine natürliche Zahl t und Äquivalenzklassen $[z_1], [z_2], \ldots, [z_t] \in \overline{X}$ gibt, sodass ein Pr-Zyklus der Art

$$[z_1]R'[z_2]R' \ldots R'[z_t]P(R')[z_1]$$

existiert. Übersetzt in die ursprüngliche Menge X und Relation R, ergibt sich daraus eine Kette der Art:

$$z_1' P(R) z_2' I(\overline{R}) z_2'' P(R) z_3' \ldots P(R) z_t' I(\overline{R}) z_t'' P(R) z_1''$$

Dabei existiert zwischen den Elementen unterschiedlicher Äquivalenzklassen eine asymmetrische Präferenz und zwischen Elementen der selben Äquivalenzklasse eine symmetrische Präferenz, die allerdings nur in der transitiven Hülle sichergestellt ist. Da die Elemente innerhalb einer Äquivalenzklasse äquivalent sind, gilt für drei Elemente der Äquivalenzklasse $[z_1]$ sowohl $z_1 I(\overline{R}) z_1'$ als auch $z_1'' I(\overline{R}) z_1$. Durch Anwendung der Transitivität von R kann dann folgende Kette geschlossen werden:

$$z_1 P(R) z_2' P(R) \ldots P(R) z_t'' P(R) z_1$$

So resultiert durch die Transitivität auch $z_1 P(R) z_1$. Zusammen mit der Feststellung, dass aus $z_1 I(\overline{R}) z_1'$ und $z_1'' I(\overline{R}) z_1$ durch die Transitivität von $\overline{R}$ $z_1 I(\overline{R}) z_1$ geschlossen werden kann, widerspricht dies der Aussage aus Andrikopoulos' Proposition 1, die besagt, dass für eine konsistente Relation R der asymmetrische Teil $P(R)$ in $P(\overline{R})$ enthalten sein muss. Somit ist die Annahme eines solchen Zyklus falsch und R' muss Δ-konsistent sein.

Da nun bewiesen ist, dass R' Δ-konsistent ist, garantiert Theorem 3.4, welches wie auch Lemma 3.2, auf das Szpilrajn-Theorem zurückgeht, die Existenz einer linearen Ordnungserweiterung von R'.

Theorem 3.4. *Sei R eine binäre Relation auf X und seien x, y zwei beliebige, nicht vergleichbare Elemente bezüglich R. Dann besitzt R eine lineare Ordnungserweiterung R^*, in der xR^*y gilt, und eine lineare Ordnungserweiterung R^{**}, in der $yR^{**}x$ gilt, genau dann, wenn R Δ-konsistent ist.*

Dieses Theorem garantiert, dass für jede Δ-konsistente Relation und für jedes Paar nicht vergleichbarer Elemente eine lineare Ordnungserweiterung existiert, in der die beiden Elemente in einer gewünschten Reihenfolge stehen [24]. Theorem 3.4 zeigt explizit, dass es für zwei nicht vergleichbare Elemente immer beide Möglichkeiten der Anordnung gibt: Es existiert sowohl eine lineare Ordnungserweiterung, in der das erste Element vor dem zweiten steht, als auch eine, in der das zweite vor dem ersten steht.

Der Beweis von Theorem 3.4 basiert, wie auch der Beweis von Lemma 3.2, auf dem Zornschen Lemma, einem grundlegenden Resultat der Mengenlehre, das besagt, dass jede partiell geordnete Menge, in der jede Kette eine obere Schranke besitzt, mindestens ein maximales Element enthält [28]. Im Kontext der Ordnungserweiterungen bedeutet dies, dass man die Menge aller konsistenten Erweiterungen von R betrachtet, diese partiell durch Inklusion ordnet und dann das Zornsche Lemma anwendet, um die Existenz einer maximalen Erweiterung zu beweisen. Formal wird die Menge aller Δ-konsistenten Erweiterungen von R, die die Präferenz (x, y) enthalten, als

$$\tilde{R} = \{\tilde{R}_i | i \in I\}$$

definiert. Für jede Kette $Q = (Q_i)i \in I$ in $\tilde{R}$ ist die Vereinigung

$$\tilde{Q} = \bigcup_{i \in I} Q_i$$

wieder eine Δ-konsistente Erweiterung von R, die (x, y) enthält. Durch das Zornsche Lemma existiert dann ein maximales Element $\hat{R} \in \tilde{R}$, welches eine transitive und totale Relation ist, die alle ursprünglichen Präferenzen und Äquivalenzen respektiert. Die Maximalität von $\hat{R}$ garantiert, dass keine weiteren Präferenzen hinzugefügt werden können, ohne die Konsistenz zu verletzen, sodass $\hat{R}$ eine lineare Ordnung darstellt.

Die so gewonnene lineare Ordnungserweiterung auf den Äquivalenzklassen $\hat{R}$ wird schließlich auf die ursprüngliche Menge zurück projiziert. Dabei gilt, dass zwei Elemente genau dann in der neuen Relation in Beziehung zueinander stehen, wenn sie entweder in der ursprünglichen Relation äquivalent sind oder wenn ihre Äquivalenzklassen in der linearen Ordnungserweiterung $\hat{R}$ in Beziehung zueinander stehen. Formal wird die resultierende Relation R^* auf X definiert durch

$$(x, y) \in R^* \Leftrightarrow (x, y) \in I(\overline{R}) \vee ([x], [y]) \in \hat{R}$$

Die resultierende Relation ist reflexiv, transitiv und total und damit eine totale Ordnungserweiterung von R. Äquivalenzen innerhalb der Äquivalenzklassen bleiben dabei erhalten, wodurch die Relation beim Auflösen der Äquivalenzklassen im Allgemeinen keine lineare Ordnungserweiterung bleibt.

Obwohl Andrikopoulos' Ansatz durch die Bildung von Äquivalenzklassen und die Reduktion auf eine Δ-konsistente Relation eine systematische Strukturierung bietet, ist die Methode in ihrer Gesamtheit nicht konstruktiv. Der Beweis stützt sich, wie auch Suzumuras Beweis, entscheidend auf das Zornsche Lemma, um die Existenz einer maximalen Erweiterung zu garantieren. Dieses Lemma liefert jedoch nur einen abstrakten Existenzbeweis ohne konkrete Konstruktionsvorschrift. insbesondere bei der Anwendung auf endliche Mengen oder in algorithmischen Kontexten ist dieser Ansatz ungeeignet, da er lediglich die Existenz einer totalen Ordnungserweiterung beweist, aber keine effektive Konstruktionsvorschrift liefert. Zudem weist das Vorgehen durch die Konstruktion aller Δ-konsistenten Erweiterungen im Worst Case eine exponentielle Komplexität auf [6]. Außerdem liefert die Methode kein Verfahren zur schrittweisen Ergänzung fehlender Präferenzen, ohne die Suzumura-Konsistenz zu verletzen.

Zusammenfassend lässt sich sagen, dass Andrikopoulos' Methode theoretisch elegant ist, aber für die konkrete Konstruktion totaler Ordnungserweiterungen unbrauchbar. Sie eignet sich weder für die Implementierung in Algorithmen noch für die schrittweise Nachvollziehbarkeit in Anwendungen.

Beispiel

Nun wird der Beweis von Andrikopoulos am in Kapitel 3.1 vorgestellten Beispiel Schritt für Schritt angewendet. Die betrachtete Menge X und Relation R sind:

$$X = \{a, b, c, d, e\}$$
$$R = \{(a, b), (b, c), (c, d), (d, c)\}$$

Im ersten Schritt werden die Äquivalenzklassen der Elemente auf Basis von $I(\overline{R} \cup \Delta)$, wobei $\overline{R}$ die transitive Hülle von R und Δ die diagonale Relation darstellt. Hierfür müssen analog zum Beweis von Suzumura zuerst sowohl die transitive Hülle von R als auch die diagonale Relation konstruiert werden.

Wie bereits in der Veranschaulichung von Suzumuras Beweis gezeigt wurde, ergeben sich folgende transitive Hülle $\overline{R}$, diagonale Relation Δ und deren Vereinigung $\overline{R} \cup \Delta$:

$$\overline{R} = \{(a, b), (b, c), (c, d), (d, c), (a, c), (b, d), (c, c), (d, d), (a, d)\}$$
$$\Delta = \{(a, a), (b, b), (c, c), (d, d), (e, e)\}$$

$$\begin{aligned}
&\overline{R} \cup \Delta \\
&= \{(a, b), (b, c), (c, d), (d, c), (a, c), (b, d), (c, c), (d, d), (a, d)\} \\
&\cup \{(a, a), (b, b), (c, c), (d, d), (e, e)\} \\
&= \{(a, b), (b, c), (c, d), (d, c), (a, c), (b, d), (c, c), (d, d), (a, d), (a, a), (b, b), (e, e)\}
\end{aligned}$$

Extrahiert man hieraus alle Präferenzen $x, y \in (\overline{R} \cup \Delta)$ für die auch $y, x \in (\overline{R} \cup \Delta)$ gilt, resultiert der symmetrische Teil der Relation $(\overline{R} \cup \Delta)$:

$$I(\overline{R} \cup \Delta) = \{(a, a), (b, b), (c, c), (d, d), (e, e), (c, d), (d, c)\}$$

Daran ist deutlich zu erkennen, dass die beiden Elemente c und d äquivalent sind, wodurch sie in derselben Äquivalenzklasse liegen. Die drei anderen Elemente bilden jeweils ihre eigene Äquivalenzklasse:

$$[a] = \{a\}, [b] = \{b\}, [c] = [d] = \{c, d\}, [e] = \{e\}$$

Daraus resultiert die Menge der Äquivalenzklassen $\overline{X}$, in der für gleiche Äquivalenzklassen, wie hier $[c]$ und $[d]$, nur eine enthalten ist:

$$\overline{X} = \{[a], [b], [c], [e]\}$$

Aufbauend auf der konstruierten Menge der Äquivalenzklassen $\overline{X}$ wird die Quotientenrelation R' gebildet. Dabei gilt:

$$([x], [y]) \in R' \Leftrightarrow \exists x' \in [x], y' \in [y] : (x', y') \in P(R)$$

Das bedeutet, dass für alle Präferenzen, die im asymmetrischen Teil von R enthalten sind, eine entsprechende Präferenz in der Quotientenrelation R' hinzugefügt wird. Hierfür ist die Bildung des asymmetrischen Teils der Relation R notwendig, der aus allen Präferenzen $x, y \in R$ besteht, für die $y, x \notin R$ gilt:

$$P(R) = \{(a, b), (b, c)\}$$

Auf Basis dessen wird die Quotientenrelation R' folgendermaßen konstruiert:

$$([a], [b]) \in R' \Leftrightarrow (a, b) \in P(R)$$
$$([b], [c]) \in R' \Leftrightarrow (b, c) \in P(R)$$

$$\Rightarrow R' = \{([a], [b]), ([b], [c])\}$$

Im nächsten Schritt wird Δ-Konsistenz von R' geprüft. Das bedeutet, dass R' Suzumura-konsistent sein und $\mathrm{I}(\overline{R}') \subseteq \Delta$ gelten muss. Da R' nur zwei Präferenzen enthält, kann sowohl die Existenz von PR-Zyklen als auch von äquivalenten Elementen schnell ausgeschlossen werden. Somit ist die Δ-Konsistenz von R' bewiesen und die Voraussetzung für die Anwendung von Theorem 3.4 erfüllt.

Dieses beweist die Existenz einer linearen Ordnungserweiterung $\hat{R}$ für die Δ-konsistente Relation R' und beginnt im ersten Schritt damit, in der Relation R'' R' mit der diagonalen Relation Δ und einer beliebigen Präferenz zweier in R' nicht vergleichbarer Elemente zu vereinigen. In diesem Fall wurde sich für die Präferenz $([a], [e])$ entschieden, da nur $[e]$ (transitiv) mit keinem der anderen Elemente vergleichbar ist und der Vergleich zu $[a]$ dazu führt, dass durch die im kommenden Schritt konstruierte transitive Hülle noch keine totale Relation entsteht.

$$R'' = R' \cup \Delta \cup \{([a], [e])\}$$
$$= \{([a], [b]), ([b], [c]), ([a], [a]), ([b], [b]), ([c], [c]), ([e], [e]), ([a], [e])\}$$

Als Nächstes wird, wie bereits angedeutet, die transitive Hülle von R'' konstruiert, wobei in jeder Iteration untersucht wird, ob es Elemente $x, y, z \in \overline{X}$ gibt, für die $(x, y), (y, z) \in R''$ gilt, aber $(x, z) \notin R''$. Ist dies der Fall, wird (x, z) der transitiven Hülle hinzugefügt.

In diesem Fall gibt es nur eine Iteration, in der folgende Präferenz hinzugefügt wird:

$$([a], [b]) \in R'' und ([b], [c]) \in R'' \Rightarrow ([a], [c]) \in \overline{R''}$$

Somit resultiert die transitive Hülle $\overline{R''}$:

$$\overline{R''} = \{([a], [b]), ([b], [c]), ([a], [a]), ([b], [b]), ([c], [c]), ([e], [e]), ([a], [e]), ([a], [c])\}$$

Darauf aufbauend wird die Menge der Δ-konsistenten Erweiterungen von $\overline{R''}$ konstruiert, welche als $\tilde{R}$ bezeichnet und folgendermaßen definiert wird:

$$\tilde{R} = \{\tilde{R}_n \subseteq X \times X | \tilde{R}_n \ ist \ \Delta - konsistent, \ \overline{R''} \subseteq \tilde{R}_n\}$$

Innerhalb von $\tilde{R}$ werden die einzelnen Mengen $\tilde{R}_n$ durch Mengeninklusion sortiert. Das bedeutet, eine Menge ist Teilmenge ihres Nachfolgers in $\tilde{R}$. Da sich aus der vollständigen Darstellung von $\tilde{R}$ keinen Mehrwert für das weitere Vorgehen bietet, wird im Rahmen dieser Arbeit nur eine Folge von $\overline{R''}$ zu einer totalen Δ-konsistenten Erweiterung abgebildet.

Das erste Element von $\tilde{R}$ - hier als $\tilde{R}_1$ bezeichnet - ist $\overline{R''}$, da jede Relation auch eine Teil-Relation ihrer selbst darstellt.

$$\tilde{R}_1 = \overline{R''} = \{([a],[b]),([b],[c]),([a],[a]),([b],[b]),([c],[c]),([e],[e]),([a],[e]),([a],[c])\}$$

Bei der Konstruktion des nächsten Elements in $\tilde{R}$ - $\tilde{R}_2$ - hat man mehrere mögliche noch nicht vergleichbare Elemente, die man $\tilde{R}_1$ hinzufügen kann. Hier wurde sich für die Präferenz $([c],[e])$ entschieden, um für die Bildung von $\tilde{R}_3$ durch Nutzung der Transitivität die Präferenz $([b],[e])$ hinzuzufügen.

$$\begin{aligned}\tilde{R}_2 &= \tilde{R}_1 \cup \{([c],[e])\} \\ &= \{([a],[b]),([b],[c]),([a],[a]),([b],[b]),([c],[c]),([e],[e]),([a],[e]),([a],[c]),([c],[e])\}\end{aligned}$$

$$\begin{aligned}\tilde{R}_3 &= \tilde{R}_2 \cup \{([b],[e])\} \\ &= \{([a],[b]),([b],[c]),([a],[a]),([b],[b]),([c],[c]),([e],[e]),([a],[e]),([a],[c]),([c],[e]), \\ &([b],[e])\}\end{aligned}$$

$\tilde{R}_3$ ist bereits total und kann somit nicht mehr durch das Hinzufügen einer weiteren Präferenz erweitert werden.

Mithilfe des Zornschen Lemmas wird nun die maximale Δ-konsistente Erweiterung $\hat{R}$ von $\overline{R''}$ bestimmt. Das Zornsche Lemma besagt, dass jede geordnete Menge, also auch $\tilde{R}$ mindestens ein maximales Element besitzt. Da $\tilde{R}_3$ total ist, bildet diese Relation eines der Maxima der Menge $\tilde{R}$ und somit gilt $\hat{R} = \tilde{R}_3$. $\hat{R}$ bildet auch direkt eine lineare Ordnungserweiterung von $\overline{R''}$.

Im nächsten Schritt werden die Äquivalenzklassen aufgelöst, wodurch aus $\hat{R}$ eine totale Ordnungserweiterung von R - hier als R^* bezeichnet - gebildet wird. Dabei gilt für die Konstruktion von R^*

$$(x,y) \in R^* \Leftrightarrow (x,y) \in I(\overline{R}) \vee ([x],[y]) \in \hat{R}$$

R^* ist also die Vereinigung von $I(\overline{R})$ mit der Relation, die sich durch die Auflösung der Äquivalenzklassen aus $\hat{R}$ bildet. Dazu wird zuerst $I(\overline{R})$ gebildet.

$$\begin{gathered}\overline{R} = \{(a,b),(b,c),(c,d),(d,c),(a,c),(b,d),(c,c),(d,d),(a,d)\} \\ \Rightarrow I(\overline{R} = \{(c,c),(c,d),(d,c),(d,d)\}\end{gathered}$$

Hinzu kommen alle Präferenzen, die sich aus $\hat{R}$ ergeben, wenn man die Äquivalenzklassen auflöst. Dabei werden die Paare aus $I(\overline{R})$ nicht nochmals angegeben, wobei sich alle vier Präferenzen aus $([c],[c])$ ergeben würden.

$$
\begin{aligned}
(a,a) \in R^* &\Leftrightarrow ([a],[a]) \in \hat{R} \\
(a,b) \in R^* &\Leftrightarrow ([a],[b]) \in \hat{R} \\
(a,c),(a,d) \in R^* &\Leftrightarrow ([a],[c]) \in \hat{R} \\
(a,e) \in R^* &\Leftrightarrow ([a],[e]) \in \hat{R} \\
(b,b) \in R^* &\Leftrightarrow ([b],[b]) \in \hat{R} \\
(b,c),(b,d) \in R^* &\Leftrightarrow ([b],[c]) \in \hat{R} \\
(b,e) \in R^* &\Leftrightarrow ([b],[e]) \in \hat{R} \\
(c,e),(d,e) \in R^* &\Leftrightarrow ([c],[e]) \in \hat{R} \\
(e,e) \in R^* &\Leftrightarrow ([e],[e]) \in \hat{R}
\end{aligned}
$$

Damit resultiert die Relation R^*:

$$
\begin{aligned}
R^* = \{&(a,a),(b,b),(c,c),(d,d),(e,e),(a,b),(a,c),(a,d),(a,e),(b,c),(b,d),(b,e),\\
&(c,d),(c,e),(d,c),(d,e)\}
\end{aligned}
$$

R^* ist sowohl eine Erweiterung der Relation R als auch vollständig, reflexiv und transitiv, aber nicht antisymmetrisch, und bildet somit eine totale Ordnungserweiterung von R.

3.4 Topologisches Sortieren

Zuletzt wird das topologische Sortieren als Möglichkeit untersucht, eine totale Ortungserweiterung aus einer Suzumura-konsistenten Relation zu konstruieren. Dies ist ein grundlegendes Verfahren der Graphentheorie und Informatik, das dazu dient, aus einem gegebenen gerichteten azyklischen Graphen (englisch: directed acyclic graph, kurz DAG) auf einer endlichen Menge von Knoten eine lineare Ordnung zu konstruieren [9]. Entsprechend kann der Algorithmus auch angewendet werden, um für eine gegebene azyklische Relation eine lineare Ordnungserweiterung zu konstruieren. Dabei bedeutet azyklisch automatisch auch antisymmetrisch, da bereits zwei äquivalente Elemente einen Zyklus darstellen.

Für die Implementierung des topologischen Sortierens gibt es zwei bekannte Ansätze. Ein Ansatz, der auch als DFS-Ansatz bekannt ist, basiert auf einer Tiefensuche durch den Graphen und arbeitet rekursiv [9]. Der zweite Ansatz - auch unter Kahns Algorithmus bekannt - basiert auf einer Breitensuche und arbeitet iterativ [16]. Da sich Kahns Algorithmus für die in Kapitel 4 noch beschriebenen Anpassungen besser eignet, wird auch an dieser Stelle nur Kahns Algorithmus beschrieben und für das DFS-Verfahren auf die Literatur verwiesen.

Kahns Algorithmus weist zu Beginn jedem Knoten die Anzahl seiner eingehenden Kanten - den Eingangsgrad - als Wert zu. Dann beginnt die erste Iteration, in der alle Knoten mit einem Eingangsgrad von null betrachtet werden. Es wird ein beliebiger Knoten mit einem Eingangsgrad von null gewählt, der linearen Ordnung hinzugefügt und samt all seiner eingehenden und ausgehenden Kanten aus dem Graphen entfernt. Im Anschluss werden die Eingangsgerade aller Knoten, für die es eine eingehende Kante vom gelöschten Knoten aus gab, um eins reduziert und die nächste Iteration beginnt.

Der Algorithmus endet, wenn keine Knoten mehr verfügbar sind oder wenn der Graph noch Knoten beinhaltet, es aber keine Knoten mehr mit einem Eingangsgrad von null gibt. Der zweite Fall kann jedoch nur auftreten, wenn der Eingangs-Graph Zyklen enthält, was durch die Voraussetzung des Verfahrens nicht erlaubt ist. So kann der Algorithmus nur enden, wenn es keine Knoten mehr zu betrachten gibt, was bedeutet, dass der Algorithmus erfolgreich eine lineare Ordnung konstruiert hat. Diese Funktionsweise ist in Algorithmus 1 dargestellt.

Input: Ein gerichteter azyklischer Graph $G = (V, E)$
Output: Eine Topologische Sortierung der Knoten in V

$L \leftarrow [\,]$; `// Liste für das Ergebnis`
$S \leftarrow \{\, v \in V \mid \deg^-(v) = 0\}$; `// Menge der Knoten ohne eingehende Kanten`
while $S \neq \varnothing$ **do**
  wähle und entferne einen Knoten n aus S;
  füge n ans Ende von L an;
  foreach *Kante* $(n, m) \in E$ **do**
    entferne Kante (n, m) aus E;
    reduziere $\deg^-(m)$ um 1;
    if $\deg^-(m) = 0$ **then**
      füge m zu S hinzu;
    end
  end
end
if $E \neq \varnothing$ **then**
  return Error: Der Graph enthält einen Zyklus;
end
return L;

Algorithmus 1: Kahns Topologisches Sortieren

Die direkte Anwendung des topologischen Sortierens ist jedoch nicht ohne Weiteres auf Suzumura-konsistente Relationen übertragbar. Der Grund liegt darin, dass die Suzumura-Konsistenz zwar Zyklen mit mindestens einer strikten Präferenz verbietet, aber äquivalente Elemente zulässt, die in Graphen zu Zyklen der Länge zwei

führen, die die Voraussetzung des topologischen Sortierens verletzen.

Beispiel

Betrachtet man das Beispiel

$$X = \{a, b, c, d, e\}$$
$$R = \{(a, b), (b, c), (c, d), (d, c)\}$$

sieht man, dass die beiden Elemente c und d äquivalent sind und damit einen Zyklus der Länge zwei bilden, was die Anwendung des topologischen Sortierens verbietet.

Grundsätzlich ist das topologische Sortieren ein effizientes Verfahren zur Konstruktion linearer Ordnungserweiterungen für azyklische Relationen. Seine Stärke liegt in der klaren algorithmischen Umsetzbarkeit und der linearen Laufzeit von $O(n+m)$. Für Suzumura-konsistente Relationen ist er jedoch ungeeignet, da äquivalente Elemente Zyklen der Länge zwei erzeugen, die die Azyklizitäts-Voraussetzung verletzen.

4 Algorithmische Konstruktion der totalen Ordnungserweiterungen

Kapitel 3 hat eindeutig gezeigt, dass sowohl die Beweise der Existenz einer totalen Ordnungserweiterung als auch das topologische Sortieren zur Konstruktion einer linearen Erweiterung nicht geeignet sind, um totale Ordnungserweiterungen einer Suzumura-konsistenten Relation zu bilden. Aus diesem Grund wird in diesem Kapitel ein Verfahren erarbeitet, welches genau dies in effizienter Laufzeit schaffen soll und zudem nicht nur eine sondern alle möglichen totalen Ordnungserweiterungen einer Suzumura-konsistenten Relation konstruiert. Im Anschluss wird die praktische Umsetzung des entwickelten Verfahrens zur Ermittlung aller totalen Ordnungserweiterungen einer Suzumura-konsistenten Relation beschrieben, welche sowohl eine durchdachte Softwarearchitektur als auch eine effiziente Implementierung des Verfahrens benötigt. Abschließend wird das entwickelte Verfahren bzgl. seiner Laufzeit analysiert und am bekannten Beispielfall veranschaulicht.

4.1 Entwicklung eines Konstruktionsverfahrens

Nach der vergleichenden Analyse bestehender Verfahren in Kapitel 3 – insbesondere der reinen Existenz-Beweise von Suzumura und Andrikopoulos sowie des klassischen topologischen Sortierens – liegt der Bedarf an einem konstruktiven, schrittweisen Verfahren offen. Ziel ist es, aus einer gegebenen Suzumura-konsistenten Relation R auf einer endlichen Menge X alle totale Ordnungserweiterungen zu er-

zeugen. Das entwickelte Verfahren kombiniert geeignete Aspekte der vorgestellten Beweise und Verfahren und wird durch geeignete Anpassungen ergänzt.

Gegeben sei eine Suzumura-konsistente Relation R auf einer Menge X. Der erste Schritt besteht - analog dem Beweisverfahren von Andrikopoulos - darin, die Elemente in Äquivalenzklassen aufzuteilen. Dabei gilt, dass zwei Elemente in einer Äquivalenzklasse abgebildet werden, wenn sie in der reflexiv-transitiven Hülle von R symmetrisch zueinander stehen. Dabei gilt

$$[x] = \{y \in X | (x, y) \in I(\overline{R} \cup \Delta)\},$$

wobei $\overline{R}$ die transitive Hülle von R und Δ die diagonale Relation auf Basis der Menge X darstellt. Basierend auf den Äquivalenzklassen wird eine Quotientenrelation R' konstruiert, wobei zwei Äquivalenzklassen in R' in Relation zueinander stehen, wenn eines oder mehrere ihrer Elemente im asymmetrischen Teil von R in Relation zueinander stehen:

$$([x], [y]) \in R' \Leftrightarrow \exists x' \in [x], y' \in [y] \ \ mit \ \ (x', y') \in P(R)$$

Dies hat den Vorteil, dass man zum einen die Komplexität der Relation reduziert und zum anderen vorhandene Zyklen beseitigt, bevor man mit den nächsten Schritten fortfährt.

Auf der Quotientenrelation R' wird ein abgewandelter Kahn-Algorithmus angewendet. Durch die Beseitigung der in Suzumura-konsistenten Relationen möglichen Zyklen im Sinne von Äquivalenzen ist die Voraussetzung für die Anwendung des topologischen Sortierens sichergestellt. In dieser Arbeit wird Kahns Algorithmus allerdings nicht verwendet, um eine lineare Ordnungserweiterungen von R' zu erzeugen.

Der Algorithmus wird so angepasst, dass er in R' zwei nicht vergleichbare Elemente a und b, nicht nur in die beiden möglichen Reihenfolgen $x \rightarrow y$ und $y \rightarrow x$ bringt, sondern auch die Möglichkeit der äquivalenten Positionierung der beiden Elemente $x \sim y$ abbildet. Diese Anpassung des Algorithmus ist notwendig, da zwei nicht vergleichbare Elemente in einer totalen Ordnungserweiterung auch äquivalent positioniert werden dürfen, Kahns Algorithmus dies aber ohne Anpassung nicht abbilden würde und so erlaubte totale Ordnungserweiterungen nicht konstruiert werden würden. Zudem wird der Algorithmus durch eine Schleife ergänzt, mithilfe der nicht nur eine totale Ordnungserweiterung erzeugt wird, sondern alle möglichen totalen Ordnungserweiterungen.

Im letzten Schritt werden für jede konstruierte totale Ordnungserweiterung der Relation R' die gebildeten Äquivalenzklassen wieder aufgelöst, wodurch die totalen Ordnungserweiterungen zur Eingangs-Relation R gebildet werden.

Beispiel

Betrachtet man das Beispiel

$$X = \{a, b, c, d, e\}$$
$$R = \{(a, b), (b, c), (c, d), (d, c)\},$$

welches in Abbildung 1 veranschaulicht ist, resultieren, wie man bei der Anwendung von Andrikopoulos' Beweisverfahren für diesen Beispielfall bereits gesehen hat, die Äquivalenzklassen

$$[a] = \{a\}, [b] = \{b\}, [c] = [d] = \{c, d\}, [e] = \{e\}$$

und die darauf basierende Quotientenrelation R'

$$R' = \{([a], [b]), ([b], [c])\},$$

die zur Veranschaulichung in Abbildung 2 dargestellt wird.

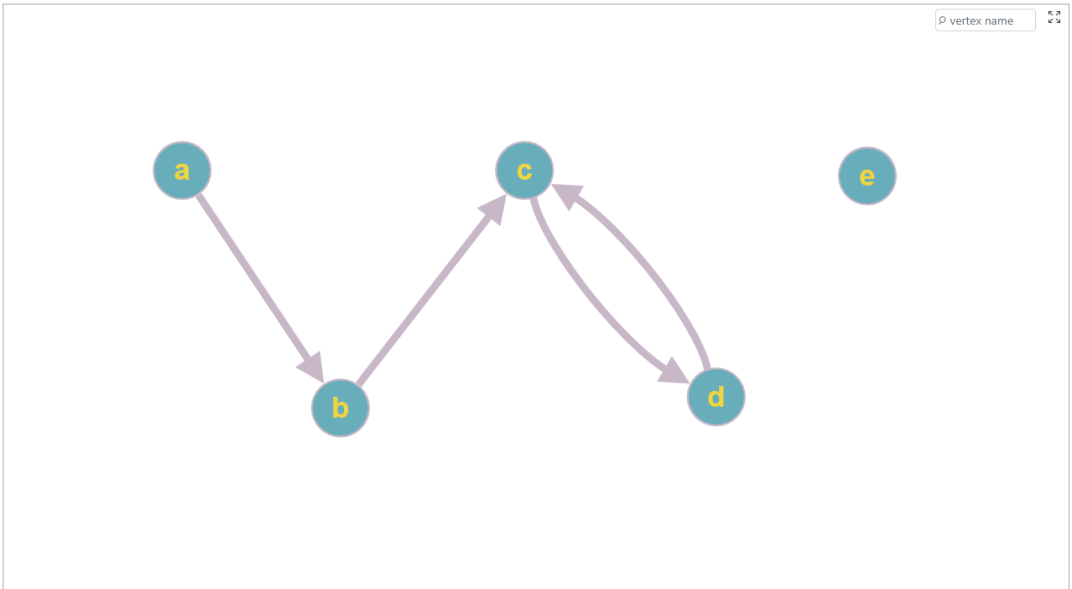

Abbildung 1: Beispiel-Relation R
Erstellt mit Graph Online, `https://graphonline.top/de/`

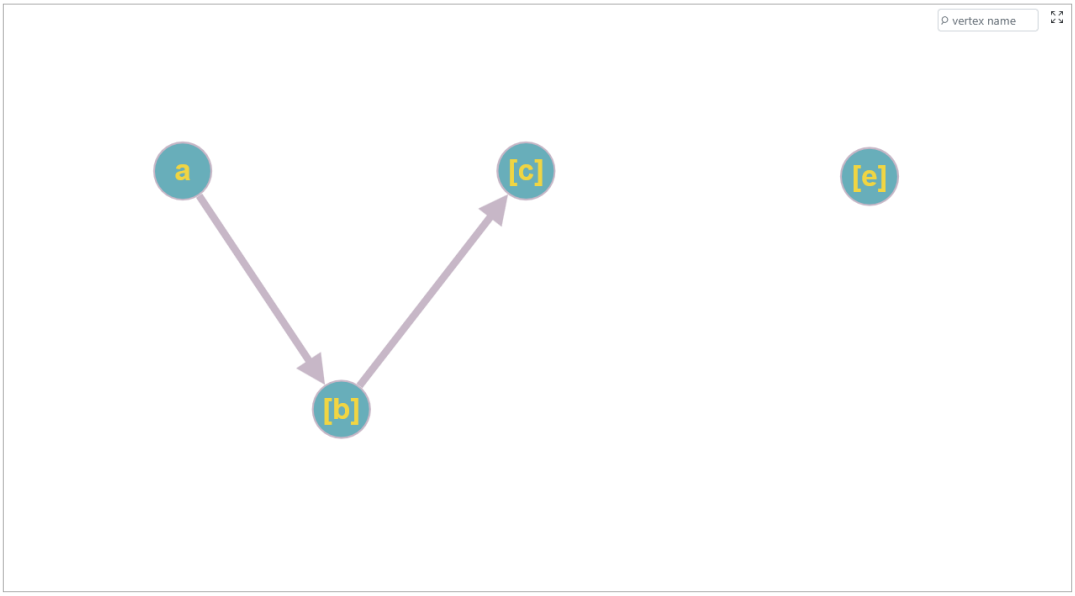

Abbildung 2: Beispiel-Quotientenrelation R'
Erstellt mit Graph Online, `https://graphonline.top/de/`

Wendet man hierauf den angepassten Kahn-Algorithmus an, erhält man folgende sieben totalen Ordnungserweiterungen, auf Basis der Quotientenrelation R':

$$
\begin{aligned}
R'_1 &= [e] \to [a] \to [b] \to [c] \\
R'_2 &= [e] \sim [a] \to [b] \to [c] \\
R'_3 &= [a] \to [e] \to [b] \to [c] \\
R'_4 &= [a] \to [e] \sim [b] \to [c] \\
R'_5 &= [a] \to [b] \to [e] \to [c] \\
R'_6 &= [a] \to [b] \to [e] \sim [c] \\
R'_7 &= [a] \to [b] \to [c] \to [e]
\end{aligned}
$$

Löst man die Äquivalenzklassen auf und berücksichtigt die zum Teil äquivalente Positionierung des „freien“ Elements e bzw. $[e]$, ergeben sich folgende sieben totale

Ordnungserweiterungen der Relation R:

$$
\begin{aligned}
R_1^* &= e \to a \to b \to \{c, d\} \\
R_2^* &= \{a, e\} \to b \to \{c, d\} \\
R_3^* &= a \to e \to b \to \{c, d\} \\
R_4^* &= a \to \{b, e\} \to \{c, d\} \\
R_5^* &= a \to b \to e \to \{c, d\} \\
R_6^* &= a \to b \to \{c, d, e\} \\
R_7^* &= a \to b \to \{c, d\} \to e
\end{aligned}
$$

An diesem veranschaulichenden Beispiel sieht man schnell, dass das entwickelte Verfahren alle sieben erwarteten totalen Ordnungserweiterung der Relation R konstruiert, wobei das „freie“ Element e an allen möglichen Positionen - vor, hinter und äquivalent zu allen anderen Elementen - eingefügt wurde.

4.2 Architekturentscheidung

Für die Implementierung wurden verschiedene Architekturpatterns evaluiert, um eine angemessene Balance zwischen akademischem Wert, Wartbarkeit und Komplexität zu finden. Eine einfache Struktur mit zwei bis drei Klassen wäre funktional ausreichend, würde allerdings zu wenig Möglichkeiten bieten, Kenntnisse in Softwarearchitektur zu demonstrieren und hätte ab einem gewissen Zeitpunkt zu einer unübersichtlichen Struktur geführt, die das Debuggen erschwert hätte. Andererseits wäre eine Clean Architecture für dieses Projekt zu komplex gewesen und hätte den Fokus vom eigentlichen Verfahren gelenkt. Einen guten Kompromiss zwischen guter Programmstruktur und vertretbarem Entwicklungsaufwand bieten vor allem das MVC-Pattern und die Layered Architecture.

Da der Fokus bei dieser Implementierung eher auf den implementierten Algorithmen als auf der Interaktion mit der GUI liegt, wurde sich für die Layered Architecture entschieden, die zudem das Testen, Debuggen und Warten der Software verbessert [26]. Die gewählte Vier-Schichten-Architektur umfasst einen Domain-Layer für die verwendeten Entitäten, einen Infrastructure Layer für die Algorithmen, einen Application Layer für die Workflow-Koordination und einen Presentation Layer für die GUI.

4.3 Implementierung der Kern-Algorithmen

In diesem Kapitel wird nun beschrieben, wie das in Kapitel 4.1 entwickelte Verfahren im Detail umgesetzt wird. Dazu wird dargestellt, wie Tarjans Algorithmus zur Identifizierung von SCC angepasst wird, um die Äquivalenzklassen zu bilden und

dabei die Suzumura-Konsistenz der Eingangs-Relation zu prüfen. Darauf aufbauend folgt die Darstellung der Bildung der Quotientenrelation und eine detailliertere Version des angepassten Kahn-Algorithmus.

Tarjans Algorithmus für zur Identifizierung stark zusammenhängender Komponenten

Wie bereits in Kapitel 4.1 beschrieben, beginnt die entwickelte Konstruktions-Vorschrift mit der Bildung von Äquivalenzklassen. Für diese Aufgabe wurde sich bei der Implementierung für eine angepasste Form von Tarjans Algorithmus entschieden, der 1972 von Robert Tarjan entwickelt wurde [25]. Dieser ist ein effizienter Algorithmus, der zur Bestimmung von SCC in gerichteten Graphen verwendet werden kann. Eine SCC ist eine maximale Menge von Knoten, innerhalb derer jeder Knoten von jedem anderen Knoten über gerichtete Pfade erreichbar ist. Auf Relationen bezogen entspricht eine SCC einem Zyklus, der im Gegensatz zu Suzumurakonsistenten Relationen auch strikte Relationen enthalten darf.

Tarjans Algorithmus, in seiner eigentlichen Form, basiert auf einer Tiefensuche und verwendet sog. Lowlink-Werte, um SCCs in linearer Zeit $O(n + m)$ zu identifizieren, wobei n die Anzahl der Knoten bzw. Elemente und m die Anzahl der Kanten bzw. Präferenzen darstellt. Der Algorithmus weist dabei jedem Knoten während einer Tiefensuche zwei Werte zu: einen Index, der die Entdeckungszeit darstellt, und einen Lowlink-Wert. Der Lowlink-Wert gibt den niedrigsten Index an, der von diesem Knoten bzw. Element aus erreichbar ist. Bei jedem Knoten wird geprüft, ob sein Lowlink-Wert und sein Index übereinstimmen. Ist dies der Fall, ist er Teil einer SCC. Der Algorithmus verwendet zudem einen Stack, um die Knoten der aktuellen SCC zu verfolgen, bis eine vollständige SCC identifiziert und extrahiert werden kann.

Da in diesem Fall nicht nur SCC, sondern Äquivalenzklassen innerhalb der eingegebenen Relationen gefunden werden sollen, musste der beschriebene Algorithmus angepasst bzw. ergänzt werden. Diese Ergänzung wurde nach dem vollständigen Durchlauf von Tarjans Algorithmus platziert und prüft für jede gefunden SCC, ob sie für alle Präferenzen auch das symmetrische Pendant enthält. Falls dies für eine Präferenz in der SCC nicht der Fall ist, handelt es nicht um eine Äquivalenzklasse, sondern um einen Zyklus mit mindestens einer strikten Präferenz.

Wird ein solcher Zyklus gefunden, bedeutet dies, dass die Relation nicht Suzumurakonsistent ist. So kann die Prüfung der Suzumura-Konsistenz geschickt in den ersten Schritt des Algorithmus integriert werden. Sofern dieser Algorithmus einen Zyklus mit strikter Präferenz erkennt, bricht der gesamte Algorithmus mit einer Fehlermeldung ab. Die Funktionsweise des angepassten Tarjan-Algorithmus ist in den Algorithmen 2 und 3 dargestellt.

Input: Relation $R = (X, P)$ mit Elementmenge X und Präferenzen $P \subseteq X \times X$
Output: Äquivalenzklassen $\mathcal{C}$ und Konsistenzflag c

```
index ← 0;                                  // Zähler für Tiefensuche
stack ← [];                                  // Leerer Stack für SCCs
indices ← ∅;                          // Verknüpfung: Element ↦ Index
lowlink ← ∅;                        // Verknüpfung: Element ↦ Lowlink
onStack ← ∅;                        // Verknüpfung: Element ↦ boolean
C ← [];                    // Liste der gefundenen Äquivalenzklassen
foreach x ∈ X do
    if x ∉ indices then
        strongConnect(x);
end
```

Algorithmus 2: Anngepasster Tarjan-Algorithmus mit Suzumura-Konsistenz-Prüfung - Teil 1

Erzeugung der Relation auf Basis der Äquivalenzklassen

Ist die Relation Suzumura-konsistent, wird eine Quotientenrelation auf Basis der gebildeten Äquivalenzklassen erzeugt. Hierfür werden in einem Durchlauf alle Präferenzen der Eingangs-Relation durchlaufen. Sind die beiden Elemente der Präferenz Teil von zwei unterschiedlichen Äquivalenzklassen, wird eine entsprechende Präferenz auf Basis der Äquivalenzklassen erzeugt und der Quotientenrelation hinzugefügt, sofern sie noch nicht enthalten ist. Befinden sich beide Elemente der untersuchten Präferenz in derselben Äquivalenzklasse, wird der Quotientenrelation keine Präferenz hinzugefügt.

Erweiterter Kahn-Algorithmus

Basierend darauf wird, wie bereits in Kapitel 4.1 beschrieben, ein angepasster Kahn-Algorithmus angewendet, um alle linearen Ordnungserweiterungen der konstruierten Quotientenrelation zu erzeugen.

Der klassische Kahn-Algorithmus beginnt mit der Berechnung der Eingangsgrade aller Knoten. In jeder Iteration werden alle Knoten mit einem Eingangsgrad von null ermittelt und einer dieser Knoten wird zusammen mit all seinen ein- und ausgehenden Kanten aus dem Graphen entfernt und der Ordnung hinzugefügt. Da es in Iterationen auch mehrere Knoten mit einem Eingangsgrad von null geben kann, die alle infrage kommen, um aus dem Graphen entfernt und der Ordnung hinzugefügt zu werden, ist die erzeugte lineare Ordnung nicht eindeutig.

Im Anschluss werden die Eingangsgrade der Nachfolge-Knoten des entfernten Knotens um eins reduziert und die nächste Iteration startet. Der Prozess wiederholt sich, bis entweder alle Knoten verarbeitet wurden und eine topologische Sor-

```
Function strongConnect(v):
    indices[v] ← index;
    lowlink[v] ← index;
    index ← index + 1;
    put v on stack;
    onStack[v] ← true;
    foreach (v, w) ∈ P do
        if w ∉ indices then
            strongConnect(w);
            lowlink[v] ← min(lowlink[v], lowlink[w]);
        else if onStack[w] then
            lowlink[v] ← min(lowlink[v], indices[w]);
    end
    if lowlink[v] = indices[v] then
        comp ← ∅;
        repeat
            popElement ← pop from stack;
            onStack[ popElement ] ← false;
            add popElement to comp;
        until popElement = v;
        add EquivalenceClass(comp) to C;
    end

c ← true;
foreach Klasse C in C ;                          // Nur Klassen mit |C|>1
 do
    if |C| > 1 then
        foreach (x, y) ∈ P mit x, y ∈ C do
            if (y, x) ∉ P then
                c ← false;
                return (C, c);          // Asymmetrischer Zyklus gefunden
            end
        end
    end
end
return (C, c);                  // Äquivalenzklassen und Konsistenzflag
```

Algorithmus 3: Angepasster Tarjan-Algorithmus mit Suzumura-Konsistenz-Prüfung - Teil 2

tierung erzeugt wurde oder nur noch Knoten mit einem Eingangsgrad größer null übrig sind, was auf einen Zyklus in der Eingangs-Relation hinweist. Der zweite Fall ist hier allerdings durch die vorherige Suzumura-Konsistenz-Prüfung und Äquivalenzklassen-Bildung ausgeschlossen.

Da der Kahn-Algorithmus in seiner Grundform allerdings nur eine lineare Ordnungserweiterung des Graphen oder der Relation erzeugt, müssen zwei Anpassungen erfolgen, da zum einen alle und zum anderen totale und nicht lineare Ordnungserweiterungen erzeugt werden sollen.

Die Konstruktion aller Ordnungserweiterungen wird durch die Implementierung einer Schleife gelöst, die immer dort ausgelöst wird, wo es mehrere Knoten bzw. Elemente mit einem Eingangsgrad von null gibt. In der angepassten Version des Algorithmus wird nicht ein zufälliger Knoten ausgewählt und in die nächste Iteration übergangen, sondern eine Schleife ausgelöst, die ab dieser Stelle für jeden Knoten mit Eingangsgrad null den kompletten weiteren Algorithmus ausführt.

Zudem sollen nicht nur die linearen, sondern die totalen Ordnungserweiterungen ermittelt werden, was bedeutet, dass die Forderung nach Antisymmetrie nicht mehr gilt. Sind in der Eingangs-Relation zwei Knoten bzw. Elemente nicht miteinander vergleichbar, gibt es sowohl eine gültige lineare Ordnungserweiterung in der $a \to b$ gilt, als auch eine gültige lineare Ordnungserweiterung in der $b \to a$ gilt. Da in einer totalen Ordnungserweiterung die Antisymmetrie-Voraussetzung nicht gilt, gibt es hier noch eine dritte Möglichkeit, nämlich die Äquivalenz der beiden Elemente $a \sim b$. Dies wurde gelöst, in jeder Iteration, in der es mehrere Knoten mit einem Eingangsgrad von null gibt, der Algorithmus nicht nur einen Knoten als nächsten auswählen kann, sondern jede nicht-leere Teilmenge der Menge der Knoten mit einem Eingangsgrad von null.

Im Anschluss an den erweiterten Kahn-Algorithmus werden die gebildeten Äquivalenzklassen wieder aufgelöst. Hierzu werden für die Ausgabe alle Äquivalenzklassen, die nur ein Element erhalten, in dieses Element aufgelöst. Enthält eine Äquivalenzklasse mehr als ein Element, werden alle enthaltenen Elemente innerhalb einer geschweiften Klammer zusammengefasst. Die beschriebene Funktionsweise des angepassten Kahn-Algorithmus ist in Algorithmus 4 dargestellt.

Streaming-Ausgabe zur Speicher-Optimierung

Eine weitere Optimierung der Implementierung ist die Streaming-Ausgabe der generierten totalen Ordnungserweiterungen. Anstatt alle Ergebnisse im Arbeitsspeicher zu sammeln und erst am Ende auszugeben, wird jede gefundene totale Ordnungserweiterung sofort nach ihrer Generierung ausgegeben. Diese Strategie reduziert den Speicherbedarf von exponentiell wachsend auf konstant, was besonders bei Fällen mit sehr vielen totalen Ordnungserweiterungen von entscheidender Bedeutung ist.

Input: Relation $R = (X, P)$ mit Elementmenge X und Präferenzmenge $P \subseteq X \times X$
Output: Liste L aller totalen Ordnungserweiterungen als Sequenzen von Knoten

$L \leftarrow []$; `// Liste für alle Ordnungserweiterungen`
$S \leftarrow \{ x \in X \mid \deg^-(x) = 0 \}$; `// Elemente ohne eingehende Präferenz`
`generateOrderings(`$P, \{\deg^-(x)\}, [], S$`);` `// Aufruf der rekursiven Methode`
return L;

Eingabe: P – Eingegebene Präferenzmenge
$\{\deg^-(x)\}$ – Eingangsgrade aller Element
current – Die bisher erzeugte Ordnung
S – Alle Elemente ohne eingehende Präferenz

Function `generateOrderings(`$P, \deg^-, \mathit{current}, S$`)`**:**
  if $S = \varnothing$; `// Keine Knoten ohne eingehenden Präferenz` **then**
    if $\bigcup_{\sigma \in \mathit{current}} \sigma = X$; `// Erzeugte Ordnung ist vollständig` **then**
      hänge current an L;
    end
    return;
  foreach *subset* $\subseteq S$ *with subset* $\neq \emptyset$ **do**
    merged$\leftarrow \bigcup_{x \in \text{subset}} \{x\}$; `// Neue Äquivalenzklasse`
    current$' \leftarrow$ current $\|$ [merged];
    $\deg^-{}' \leftarrow \deg^-$ kopiere;
    foreach $x \in$ *subset* **do**
      entferne x aus $\deg^-{}'$;
      foreach $(x, y) \in P$ **do**
        $\deg^-{}'(y) \leftarrow \deg^-{}'(y) - 1$;
      end
    end
    $S' \leftarrow \{ u \mid u \in \mathrm{dom}(\deg^-{}'), \deg^-{}'(u) = 0 \}$; `// Neue Menge von Elementen Eingangsgrad = 0`
    `generateOrderings(`$P, \deg^-{}', \mathit{current}', S'$`);`
  end

Algorithmus 4: Erweiterter Kahn-Algorithmus für totale Ordnungserweiterungen

4.4 Laufzeitanalyse des Konstruktionsverfahrens

Nach der Darstellung der verwendeten Algorithmen wird deren Laufzeit und die Laufzeit des entwickelten Gesamtverfahrens betrachtet, um bewerten zu können, ob eine effiziente Umsetzung möglich ist.

Komplexität pro totaler Ordnungserweiterung

Tarjans Algorithmus zur Erkennung von SCC hat in seiner Grundform eine Laufzeit von $O(n+m)$, wobei n die Anzahl der Elemente und m die Anzahl der Präferenzen angibt. Der zusätzlich eingefügte Schritt, dass innerhalb der ermittelten SCC für jede Kante geprüft wird, ob ihr symmetrisches Gegenstück ebenfalls enthalten ist, prüft im Worst Case jede Kante bzw. Präferenz der Relation und hat dementsprechend eine Laufzeit von $O(m)$ bei der Verwendung einer geeigneten Datenstruktur. Damit bleibt die Laufzeit des angepassten Tarjan-Algorithmus bei $O(n+m)$.

Im nächsten Schritt wird aus der Eingangspräferenz und den ermittelten Äquivalenzklassen eine Quotientenrelation konstruiert. Hierbei wird jede Präferenz der Eingangs-Relation betrachtet und entschieden, ob sie, angepasst an die Äquivalenzklassen, in die Quotientenrelation übernommen wird. Damit entspricht die Laufzeit dieses Algorithmus $O(m)$.

Darauf aufbauend wird der angepasste Kahn-Algorithmus angewandt, der in seiner Grundform, ebenso wie Tarjans Algorithmus, eine Laufzeit von $O(n+m)$ hat. Die beiden Anpassungen - das Einfügen der Schleife zur Betrachtung aller Möglichkeiten und die Betrachtung aller nicht-leeren Partitionen der Knoten mit einem Eingangsgrad von null - beeinflussen die Laufzeit pro gefundener totalen Ordnungserweiterung nicht, wodurch die Laufzeit bei $O(n+m)$ bleibt.

Am Ende des Algorithmus werden die Äquivalenzklassen nochmal so aufgelöst, dass in der Ordnung nicht nur die Bezeichnung der Äquivalenzklasse, sondern alle in ihr enthaltenen Elemente mit ausgegeben werden. Um eine lineare Laufzeit pro totaler Ordnungserweiterung zu gewährleisten, wird darauf verzichtet, eine vollständige Relation mit allen Präferenzen zwischen den Elementen auszugeben, die eine quadratische Laufzeit $O(n^2)$ zur Folge hätte, da für die Ergänzung der fehlenden Präferenzen alle Paare von Elementen betrachtet werden müssten. Stattdessen wird nur die sich ergebende Ordnung der Elemente ausgegeben, wobei eine Äquivalenzklasse durch eine geschweifte Klammer dargestellt wird. Somit müssen zur Auflösung der Äquivalenzklassen nur die in ihr enthaltenen Elemente berücksichtigt werden, wodurch sich für diesen Schritt eine Laufzeit von $O(n)$ ergibt.

Somit ergibt sich für die vier Schritte eine lineare Gesamtlaufzeit von $O(n+m)$. Berücksichtigt man zur Vollständigkeit noch die Verarbeitung der Eingabe und die Ausgabe, hat dies keinen Einfluss auf die lineare Gesamtlaufzeit, da die Eingabe alle Elemente und Präferenzen enthält, wodurch die Verarbeitung eine Laufzeit von

$O(n+m)$ hat. Ausgegeben wird nur die Ordnung der Elemente, wodurch diese eine Laufzeit von $O(n)$ hat.

Gesamtkomplexität des Algorithmus

Betrachtet man die Gesamtlaufzeit des Algorithmus, beträgt diese $O((n+m) \times T)$, wobei T die Anzahl der generierten totalen Ordnungserweiterung darstellt. Diese kann entgegen der Laufzeit pro Ordnungserweiterung durchaus auch exponentiell werden, da die Anzahl der möglichen totalen Ordnungserweiterungen ab einer gewissen Komplexität der Eingangs-Relation exponentiell oder sogar schneller mit der Anzahl der nicht vergleichbaren Elemente in der Eingangs-Relation wächst. Dies kann damit gezeigt werden, dass die Anzahl der möglichen totalen Ordnungserweiterungen von Relationen mit zwei nicht miteinander vergleichbaren Teil-Relationen durch die Delannoy-Zahlen angegeben werden kann, wobei die Anzahl der Äquivalenzklassen in den beiden Teil-Relationen die Eingabegrößen der Delannoy-Zahlen sind[8].

Die Delannoy-Zahlen geben grundsätzlich an, wie viele Pfade es in einem Gitter von Punkt (0, 0) zu einem Punkte (n, m) gibt, wobei senkrechte, waagerechte wie auch diagonale Schritte erlaubt sind. Die Formel der Delannoy-Zahlen lautet $D(n,m) = \sum_{k=0}^{\min(n,m)} \binom{n}{k}\binom{m}{k}2^k$ und zeigt polynomielles Verhalten. Dies kann ebenso auf die Anzahl von totalen Ordnungserweiterungen von Relationen angewandt werden, die aus zwei nicht miteinander vergleichbaren Teil-Relationen bestehen, da zwei nicht miteinander vergleichbare Elemente dabei auch auf drei Weisen zueinander positioniert werden können.

Anhand des in dieser Arbeit behandelten Beispiels wird dies nun kurz veranschaulicht und gezeigt, dass die Eingaben der Delannoy-Zahlen nicht direkt die Anzahl der Elemente der beiden Teil-Relationen sind, sondern die Anzahl der Äquivalenzklassen der beiden Teil-Relationen.

Betrachtet man das Beispiel $X = \{a, b, c, d, e\}$ und $R = \{(a, b), (b, c), (c, d), (d, c)\}$ bilden zum einen die Elemente a, b, c, d eine Teil-Relation und das Element e bildet eine Teil-Relation, die mit der anderen Teil-Relation nicht vergleichbar ist. Dabei besteht die erste Teil-Relation aus vier Elementen, aber nur 3 Äquivalenzklassen, da die beiden Elemente c und d äquivalent sind. Somit kann die Anzahl der totalen Ordnungserweiterungen der Relation R durch D(3,1) = 7 angegeben werden.

Besteht die Eingangs-Relation aus mehr als zwei nicht vergleichbarer Teil-Relationen, ist zum aktuellen Zeitpunkt keine kompakte Formel bekannt, mit der die Anzahl der möglichen totalen Ordnungserweiterungen angegeben werden kann.

Der Worst Case ist eine Eingangs-Relation mit n Elemente ohne jegliche Präferenz. Dabei kann die Anzahl der totalen Ordnungen durch die Fubini-Zahlen angegeben werden, die mit der Anzahl der Elemente n schneller als exponentiell wachsen [18].

4.5 Praktische Aspekte der Implementierung

Zum Abschluss des theoretischen Teils der Algorithmus-Entwicklung und -Darstellung wird nun noch der praktische Aspekt betrachtet. Dazu gehört zum einen die GUI und zum anderen die Validierung der Eingabedaten inkl. einer aussagekräftigen Ausgabe, um die Fehler gezielt beheben zu können.

Benutzeroberfläche

Die Implementierung umfasst zudem eine GUI, die eine intuitive Eingabe von Elementen und Präferenzen ermöglicht. Ebenso kann eine txt-Datei mit den Elementen und den Präferenzen eingelesen werden. Bei einer Eingabe über die GUI, erfolgt auch die Ausgabe per GUI. Bei dem Einlesen einer txt-Datei erfolgt die Ausgabe als txt-Datei. Die Eingabevalidierung erfolgt durch eine umfassende Fehlerprüfung, wodurch die robuste Behandlung ungültiger Eingaben gewährleistet wird. Bei erfolgreicher Verarbeitung der Eingabedaten zeigt die GUI die Ergebnisse in strukturierter Form an, aber auch eine aussagekräftige Fehlermeldung, wenn die Eingabe fehlerhaft oder nicht Suzumura-konsistent ist.

Die Eingabefelder unterstützen mathematische Notation für Elemente im Format a,b,... und Präferenzen im Format (a,b). Solange der Konstruktions-Algorithmus noch nicht gestartet wurde, wird im Ausgabebereich eine Anleitung mit Formatierungshinweisen angezeigt. Ein vorausgefülltes Beispiel (siehe Beispielfall aus Kapitel 3.1) ermöglicht sofortiges Testen ohne manuelle Eingabe.

Validierung der Eingabe

Die Implementierung enthält eine umfassende Fehlerbehandlung für verschiedene Problemklassen, von Eingabefehlern über die Verletzung der Suzumura-Konsistenz bis hin zu unerwarteten Algorithmus-Zuständen. Fehlermeldungen sind benutzerfreundlich formuliert und geben konkrete Hinweise zur Problembehebung, wie bspw. den Zyklus von Elementen, der die Suzumura-Konsistenz verletzt.

4.6 Anwendungsbeispiel

Die Demonstration des Algorithmus erfolgt, wie auch in Kapitel 3, anhand der Menge $X = \{a, b, c, d, e\}$ und der Relation $R = \{(a, b), (b, c), (c, d), (d, c)\}$. Diese Relation enthält, wie bereits dargestellt, einen symmetrischen Zyklus zwischen den Elementen c und d, was sie zu einem geeigneten Testfall für die Suzumura-Konsistenz-Prüfung macht.

Der Algorithmus beginnt mithilfe des angepassten Tarjan-Algorithmus mit der systematischen Analyse der Relation und identifiziert dabei sowohl Zyklen als auch die resultierenden Äquivalenzklassen. Die Untersuchung der Relation beginnt bei

Element a und folgt dem Pfad der Präferenzen: $a \to b \to c \to d$. An dieser Stelle wird ein entscheidender Moment erreicht, da von d aus eine Präferenz zurück zu c führt, wodurch ein Zyklus erkannt wird. Die Zykluserkennung zwischen c und d ist von zentraler Bedeutung für die Suzumura-Konsistenz-Prüfung. Der Algorithmus stellt fest, dass der Zyklus aus den Relationen (c, d) und (d, c) besteht und damit einen symmetrischer Zyklus ohne strikte Präferenz darstellt, wodurch die Suzumura-Konsistenz gewahrt bleibt.

Element e wird als isoliertes Element erkannt, da es weder eingehende noch ausgehende Präferenzen besitzt. Dies führt automatisch zu einer separaten Äquivalenzklasse mit nur einem Element. Der Algorithmus identifiziert damit vier Äquivalenzklassen: $C_1 = \{a\}, C_2 = \{b\}, C_3 = \{c, d\}$ und $C_4 = \{e\}$.

Aus den identifizierten Äquivalenzklassen entsteht eine vereinfachte Quotientenrelation mit der Struktur $C_1 \to C_2 \to C_3$, während C_4 isoliert bleibt. Diese Struktur zeigt, dass die ursprüngliche Relation auf drei verbundene Äquivalenzklassen und eine isolierte Klasse reduziert werden kann.

Die angepasste Variante von Kahns Algorithmus arbeitet in der ersten Iteration mit den verfügbaren Äquivalenzklassen C_1 und C_4, die beide keine Vorgänger und damit einen Eingangsgrad von null haben. Der Algorithmus führt eine systematische Exploration aller Kombinationsmöglichkeiten durch, wobei die rekursive Schleife alle nicht-leeren Teilmengen der verfügbaren Elemente betrachtet. Im ersten Schritt müssen dementsprechend die Auswahl-Möglichkeiten C_1, C_4 sowie C_1, C_4 betrachtet werden. Der vollständige Durchlauf des angepassten Kahn-Algorithmus führt dann zu den folgenden sieben totalen Ordnungen:

$$
\begin{aligned}
e < a < b < \{c, d\} \\
\{a, e\} < b < \{c, d\} \\
a < e < b < \{c, d\} \\
a < \{b, e\} < \{c, d\} \\
a < b < e < \{c, d\} \\
a < b < \{e, c, d\} \\
a < b < \{c, d\} < e
\end{aligned}
$$

Jede totale Ordnung wird sofort nach ihrer Generierung ausgegeben, wodurch der Speicherbedarf konstant bleibt. Die sieben generierten totalen Ordnungen repräsentieren alle zulässigen Möglichkeiten, die ursprünglich unvergleichbaren Elemente in eine totale Ordnung zu bringen, wobei die ursprünglichen strikten Präferenzen (a, b) und (b, c) sowie die Äquivalenz zwischen c und d stets erhalten bleibt.

5 Implementierung des entwickelten Konstruktionsverfahrens in Java

Nachdem in Kapitel 4 die theoretische Entwicklung des Verfahrens zur Konstruktion aller totalen Ordnungserweiterungen einer Suzumura-konsistenten Relation dargestellt wurde, behandelt dieses Kapitel die praktische Umsetzung in Java. Die Implementierung folgt den Prinzipien der Layered Architecture und demonstriert, wie das in Kapitel 4 entwickelte Verfahren in eine test-, wart- und erweiterbare Softwarearchitektur überführt werden können. Dabei werden sowohl die fachlichen Anforderungen als auch technische Aspekte wie Speichereffizienz und Benutzerfreundlichkeit berücksichtigt.

5.1 Übersicht der Implementierungsarchitektur

Die Implementierung des entwickelten Algorithmus wurde in einer Layered Architecture umgesetzt, die eine klare Trennung der Verantwortlichkeiten gewährleistet. Diese Architekturentscheidung wurde getroffen, um die Komplexität der algorithmischen Lösung angemessen zu strukturieren und gleichzeitig Wartbarkeit, Testbarkeit und Erweiterbarkeit sicherzustellen [26].

Die gewählte Architektur gliedert sich in folgende vier Schichten. Der Domain Layer dient der Modellierung der fachlichen Konzepte. Dazu gehören das Element, die Präferenz, die Äquivalenzklasse und die resultierende Ordnung. Im darüber liegenden Infrastructure Layer sind die verwendeten Kern-Algorithmen implementiert. Neben den angepassten Algorithmen von Kahn und Tarjan ist hier ebenso ein Algorithmus implementiert, der die konstruierten totalen Ordnungen für den Output in der GUI oder der txt-Datei passend formatiert. Im Application Layer findet die Koordination der Geschäftslogik der Implementierung statt. Hier ist sowohl die Algorithmus-Koordination zum Konstruieren aller totalen Ordnungserweiterungen für die eingegebene Menge und Relation als auch das Workflow-Management für die Interaktion mit der GUI implementiert. Der oberste Layer ist der Presentation Layer. Hier ist zum einen die GUI für die Ein- und Ausgabe, das Einlesen einer txt-Datei und das Starten des Algorithmus implementiert und zum anderen ein Validator, der die Eingaben vor dem Starten des Haupt-Algorithmus prüft und ggf. eine Fehlermeldung ausgibt.

Diese Struktur ermöglicht es, die Implementierung in einzelne, klar voneinander abgegrenzte Module zu unterteilen. Die Abhängigkeiten zwischen den Layern sind so aufgebaut, dass jeder Layer nur von den darunterliegenden Layern abhängt.

5.2 Domain Layer: Modellierung der fachlichen Konzepte

Der Domain Layer bildet das Fundament der Implementierung und modelliert die mathematischen Konzepte als Java-Objekte. Neben den Elementen und den Präferenzen einer Relation werden auch die Äquivalenzklassen und die erzeugten Ordnungen inkl. der notwendigen Methoden implementiert.

Grundlegende Domäne-Objekte

Die Element-Klasse repräsentiert die Elemente der Grundmenge X als unveränderliches String-Objekt und überschriebenen equals()- und hashCode()-Methoden, die die sichere Verwendung in Hash-basierten Collections ermöglichen, was der Effizienz der nachgelagerten Algorithmen dient [12]. Im Konstruktor wird dabei sichergestellt, dass nur gültige Elementnamen (nicht-null und nicht-leer) akzeptiert werden, wodurch die Datenintegrität sichergestellt wird.

Die Preference-Klasse modelliert die Präferenzen zwischen zwei Elementen durch Komposition zweier Element-Objekte. Die Klasse implementiert Operationen wie bspw. reverse() für die Umkehrung einer Präferenz, welche für die Suzumura-Konsistenz-Prüfung erforderlich ist. Die toString()-Methode ist so umgesetzt, dass sie die Präferenzen in der Notation (a,b) formatiert ausgibt.

Äquivalenzklassen und Ordnungsstrukturen

Die EquivalenceClass-Klasse stellt die komplexeste Klasse des Domain Layers dar, da sie sowohl einzelne Elemente als auch Gruppen äquivalenter Elemente repräsentieren muss. Die interne Verwendung eines HashSet<Element> verhindert automatisch Duplikate und bietet eine Containment-Prüfung in $O(1)$ Laufzeit [20]. Die Unterscheidung zwischen Singleton- und Mehrfach-Äquivalenzklassen durch die isSingleton()-Methode ermöglicht die Ausgabe entsprechend der mathematischen Konvention: Einzelne Elemente werden ohne Mengenklammern dargestellt, während Äquivalenzklassen mit mehr als einem Element in der Mengen-Notation {a,b,c} dargestellt werden.

Die Ordering-Klasse kapselt das Ergebnis einer vollständigen Ordnung als strukturiertes Objekt mit der entsprechenden Ordnung und einer Nummerierung. Die Verwendung einer HashList<EquivalenceClass> statt eines Sets ist bewusst gewählt, da die Reihenfolge der Äquivalenzklassen die Ordnung repräsentiert und somit relevant ist [19].

5.3 Infrastructure Layer: Algorithmische Kernimplementierungen

Der Infrastructure Layer bildet das Kernstück der Implementierung und enthält zum einen die beiden in Kapitel 4 entwickelten Algorithmen - den angepassten

Kahn-Algorithmus und den angepassten Tarjan-Algorithmus - und zum anderen die Klasse OutputFormatter, die die erzeugten totalen Ordnungserweiterungen für die Ausgabe geeignet formatiert.

Angepasster Tarjan-Algorithmus

Die Klasse TarjanAlgorithm enthält die Implementierung des in Kapitel 4 beschriebenen angepassten Tarjan-Algorithmus, der der Identifikation von SCC in der Eingangs-Relation dient und gleichzeitig die Suzumura-Konsistenz überprüft. Die Grundlage bildet eine Tiefensuche, die systematisch alle Elemente der Relation besucht und dabei jedem Element zwei zentrale Werte zuordnet. Zum einen den Entdeckungsindex (index), der die Reihenfolge der erstmaligen Entdeckung während der Suche abbildet und zum anderen den Lowlink-Wert (lowlink), der den niedrigsten Entdeckungsindex bezeichnet, der von diesem Element über die ausgehenden Präferenzen erreichbar ist.

Diese Werte bilden den Kern der Logik, um Wurzelelemente von SCCs zu identifizieren. Sobald bei der Rückkehr in der Rekursion ein Element entdeckt wird, dessen lowlink-Wert mit seinem index übereinstimmt, gilt dieses Element als Wurzel einer SCC. Der Algorithmus entfernt daraufhin sukzessive die Knoten dieser SCC aus einem Stack, der alle Knoten der aktuellen SCC verwaltet.

Die Erweiterung des klassischen Tarjan-Algorithmus zur gleichzeitigen Abbildung der Suzumura-Konsistenz-Prüfung wurde folgendermaßen ergänzt. Nach der Identifikation jeder SCC wird geprüft, ob alle Präferenzen innerhalb dieser SCC symmetrisch sind. Dies bedeutet, dass für jede Präferenz innerhalb der SCC auch die dazu symmetrische Präferenz in der Relation enthalten sein muss. Nur wenn diese Bedingung erfüllt ist, ist die eingegebene Relation Suzumura-konsistent. Sollte eine nicht-symmetrische (strikte) Präferenz innerhalb der SCC vorliegen, stellt diese SCC einen PR-Zyklus dar und widerspricht der Suzumura-Konsistenz. Daraufhin wird der gesamte Algorithmus unterbrochen und eine Exception aktiviert, die die Ausgabe einer aussagekräftigen Nachricht aktiviert, die auf die Suzumura-Inkonsistenz hinweist.

Die Verknüpfung der SCC-Identifizierungen mit der Suzumura-Konsistenz-Prüfung reduziert den Implementationsaufwand und erhöht die Effizienz im Vergleich zu getrennten Prüfschritten. Dadurch wurde es möglich, dass dieser Teil der Implementierung trotz der zusätzlichen Suzumura-Konsistenz-Prüfung die lineare Laufzeit des originären Tarjan-Algorithmus von $O(n + m)$ bei n Elementen und m Präferenzen beibehalten konnte, da die zusätzliche Prüfung jede Präferenz der gefundenen SCC überprüfen muss und somit eine Laufzeit von $O(m)$ hat.

Erweiterter Kahn-Algorithmus

Die Klasse KahnAlgorithm realisiert die Erweiterung des klassischen Kahn-Algorithmus für die topologische Sortierung, um die Generierung aller totalen Ordnungserweiterungen einer Suzumura-konsistenten Relation zu ermöglichen. Während Kahns Original-Algorithmus eine lineare Anordnung der Knoten in einem DAG konstruiert, umfasst die implementierte Erweiterung eine vollständige Ausgabe aller zulässigen totalen Ordnungen unter Berücksichtigung von Äquivalenzklassen.

Wie im klassischen Verfahren arbeitet der Algorithmus mit einer dynamischen Verwaltung der Eingangsgrade aller Elemente, die die Anzahl an eingehenden Präferenzen angibt. Zu Beginn werden alle Elemente mit einem Eingangsgrad von null in die Menge S aufgenommen, um sie direkt beim ersten Durchlauf verarbeiten zu können.

Der entscheidende Unterschied liegt in der rekursiven Verarbeitung aller nicht-leeren Teilmengen von S. Jede solche Teilmenge von Elementen mit einem Eingangsgrad von null bildet eine neue Äquivalenzklasse, die durch Verknüpfen der entsprechenden Elemente gebildet wird. Nach Auswahl einer Teilmenge wird eine Kopie der Eingangsgrade erstellt, die dadurch aktualisiert wird, dass alle Elemente der neu erzeugten Äquivalenzklasse aus der aktuellen Relation entfernt und die Eingangsgrade der Nachfolge-Elemente entsprechend reduziert werden.

Dieser rekursive Vorgang durchläuft somit systematisch alle möglichen Kombinationen nicht-leerer Teilmengen der verfügbaren Elemente und erzeugt alle totalen Ordnungserweiterungen. Durch diesen Ansatz werden insbesondere auch Möglichkeiten modelliert, in denen zuvor nicht vergleichbare Elemente äquivalent zueinander angeordnet werden.

Um trotz der potenziell sehr großen Menge an totalen Ordnungserweiterungen den benötigten Speicherplatz konstant zu halten, wird eine Streaming-Ausgabe eingesetzt. Hierzu wird jede gefundene totale Ordnung sofort über eine Callback-Funktion ausgegeben, wodurch nicht alle Ergebnisse bis zur Ausgabe im Arbeitsspeicher vorgehalten werden müssen. Somit bleibt der Speicherbedarf unabhängig von der Anzahl generierter Ordnungserweiterungen konstant.

Abschließend garantiert die Klasse mit diesem Verfahren eine korrekte, vollständige und speicher-effiziente Generierung aller totaler Ordnungserweiterungen einer Suzumura-konsistenten Relation.

Ausgabeformatierung und Präsentation

Die letzte Komponente des Infrastructure Layers ist der OutputFormatter. Dieser dient der Überführung von Datenstrukturen zu ihrer textuellen Repräsentation. Die

Klasse übernimmt die Konvertierung der vom Kahn-Algorithmus generierten Ordering-Objekte in benutzerfreundliche String-Repräsentationen.

Die Formatierungslogik unterscheidet zwischen verschiedenen Darstellungsformen entsprechend der Notationen. Äquivalenzklassen werden dahingehend differenziert betrachtet, da sie sowohl aus einem als auch aus mehreren Elementen bestehen können. Enthält ein EquivalenceClass-Objekt nur ein Element, wird auch nur dieses Element ausgegeben. Enthält das Objekt mehr als ein Element, wird die Äquivalenzklasse in der Mengen-Notation {a,b,c} ausgegeben, wobei die enthaltenen Elemente alphabetisch sortiert werden. Die konstruierten Ordnungserweiterungen werden so ausgegeben, dass die Elemente bzw. Äquivalenzklassen der Ordnungserweiterung durch das '<'-Symbol getrennt werden.

Der OutputFormatter unterstützt sowohl die Ausgabe in der GUI als auch in einer txt-Datei. Unterschieden wird innerhalb der Klasse nur dahingehend, dass für die Ausgabe in eine txt-Datei auch ein Header erzeugt wird, der die eingegebene Menge X und Relation R enthält.

5.4 Application Layer: Geschäftslogik und Workflow-Koordination

Der Application Layer koordiniert die Zusammenarbeit zwischen den Infrastructure-Komponenten und implementiert die übergeordnete Geschäftslogik. Dazu gehört zum einen die Koordination der Methoden zur Konstruktion der totalen Ordnungserweiterungen und zum anderen die Steuerung des gesamten Prozesses inkl. der Verarbeitung der Ein- und Formatierung der Ausgabe. Die beiden zentralen Komponenten - OrderingExtensionCalculator und OrderingProcessCoordinator - realisieren dabei sowohl die fachliche als auch die technische Workflow-Koordination.

Steuerung der fachlichen Logik

Der OrderingExtensionCalculator bildet die zentrale Fassade für alle fachlichen Operationen. Die Klasse kapselt die gesamte fachliche Logik der Ordnungserweiterungs-Berechnung und stellt eine vereinfachte Schnittstelle zu den komplexen Infrastructure-Algorithmen bereit.

Die Hauptmethode 'generateAllOrderings()' koordiniert den gesamten Workflow. Dabei wird zuerst der angepasste Algorithmus von Tarjan mit den eingegebenen Elementen und Präferenzen aufgerufen, um die Struktur und damit auch die Suzumura-Konsistenz der Eingangs-Relation zu untersuchen. Suzumura-Inkonsistenzen werden sofort nach ihrer Erkennung als 'SuzumuraInconsistentException' ausgegeben, ohne dass die aufwendige Generierung der Ordnungserweiterungen gestartet wird. Diese frühzeitige Fehlererkennung verbessert sowohl die Benutzererfahrung als auch die Performance. Ist die Eingangs-Relation Suzumura-konsistent, wird der

erweiterte Kahn-Algorithmus zur Generierung aller totalen Ordnungserweiterungen aufgerufen. Dabei ist die Streaming-Ausgabe über Consumer-Callbacks integriert, um den notwendigen Speicher konstant zu halten.

Übergeordnete Ablaufsteuerung

Während der OrderingExtensionCalculator die fachliche Logik darstellt, übernimmt die Klasse OrderingProcessCoordinator die übergeordnete Ablaufsteuerung für verschiedene Ausgabe-Möglichkeiten. Sie kümmert sich zum einen darum, Eingaben, sowohl aus der GUI als auch aus einer Datei, zu validieren und für die weitere Verarbeitung zu formatieren. Zum anderen ruft sie über den OrderingExtensionCalculator die Generierung der totalen Ordnungserweiterungen auf und protokolliert dabei über einen Consumer die konstruierten und formatierten Ordnungen, aber auch eventuelle Fehlermeldungen. Schließlich sammelt sie Kennzahlen wie die Gesamtanzahl der ermittelten Ordnungserweiterungen und die Laufzeit der Berechnung und stellt sie diese als Statistik zusammen mit den Ergebnissen für die Ausgabe zur Verfügung.

5.5 Presentation Layer: Benutzerinteraktion und Validierung

Der Presentation Layer umfasst sowohl die Darstellung der GUI, die dem Nutzer eine intuitive und bedienerfreundliche Steuerung aller Funktionen ermöglicht als auch die Validierung und Aufbereitung aller Benutzereingaben. Die Eingaben für die Elemente der Grundmenge und der Präferenzen werden auf formale Korrektheit und Konsistenz geprüft, bevor sie an die Geschäftslogik weitergeleitet werden. Durch die Kombination aus strukturiertem Layout und Validierungsmechanismen mit sofortigem Feedback sorgt der Presentation Layer für eine transparente Benutzererfahrung.

Swing-basierte Benutzeroberfläche

Die Klasse OrderingExtensionGUI realisiert die GUI der Anwendung auf Basis des Swing-Frameworks. Sie ist als ereignisgesteuerte Anwendung konzipiert, bei der Benutzerinteraktionen und Programmlogik durch Event-Handler verknüpft werden. Um sicherzustellen, dass die Oberfläche auch während längerer Berechnungen reaktionsfähig bleibt und die Streaming-Ausgabe realisiert, wird die parallele Ausführung der Algorithmus-Operationen mithilfe des SwingWorker-Mechanismus, also einem zweiten Thread, realisiert. So können die Ordnungserweiterungen im Hintergrund erzeugt werden, während der Nutzer weiterhin die bisherigen Ergebnisse einsehen kann.

Die GUI bietet übersichtliche Eingabefelder für die Menge der Elemente und die Präferenzen der Relation, die mit Tooltips und klarer Formatierung die Eingabe der

Daten erleichtern. Für umfangreichere Eingaben unterstützt die GUI den komfortablen Im- und Export von Daten über txt-Dateien. Die Ausgabe der generierten Ordnungserweiterungen erfolgt in einem scrollbaren Textbereich, der während der Konstruktion automatisch zu den neuesten Ergebnissen scrollt und so auch bei großen Ergebnismengen übersichtlich bleibt. Das Layout folgt bewährten Konzepten. So ist die Hauptkomponente mithilfe des BorderLayouts strukturiert, während die Eingabefelder mithilfe des GridBagLayouts und die Buttons durch das FlowLayout positioniert wurden.

Eingabevalidierung und Parsing

Zur Sicherstellung der Integrität der eingegebenen Daten und Vermeidung von Fehlern übernimmt der InputValidator eine zentrale Rolle. Er implementiert eine mehrstufige Validierung der eingegebenen Daten, die zu Beginn sowohl Elemente als auch Präferenzen mittels regulärer Ausdrücke prüft. Elementnamen müssen einem klar definierten Muster aus alphanumerischen Zeichen und Unterstrichen folgen und Präferenzen werden auf das korrekte Format mit Klammern und Komma geprüft, um eine einheitliche Syntax zu gewährleisten.

Nach erfolgreicher syntaktischer Prüfung folgt die semantische Validierung. Diese stellt sicher, dass alle in den Präferenzen verwendeten Elemente auch in der eingegebenen Element-Menge enthalten sind. Wird in einer Präferenz ein nicht eingegebenes Element verwendet, stoppt das Programm mit Ausgabe einer aussagekräftigen Fehlermeldung. Zusätzlich prüft der Validator, ob mindestens ein Element vorhanden ist und keine ungültigen oder leeren Präferenzen eingegeben wurden. Abschließend wandelt der Validator die Eingabestrings in Element- und Preference-Objekte um, um die Weiterverarbeitung vorzubereiten.

6 Evaluation der Implementierung

Die entwickelte Implementierung zur Konstruktion totaler Ordnungserweiterungen aus Suzumura-konsistenten Relationen wird nun abschließend systematisch evaluiert, um sowohl die funktionale Korrektheit als auch die Performance zu bewerten. Die Evaluation erfolgt anhand ausgewählter Test-Relationen, die von einfachen Beispielen zur Darstellung der qualitativen Funktionalität bis hin zu Beispielen mit sehr umfangreichen Ergebnismengen zur Darstellung von Stress-Tests reichen.

6.1 Qualitative Evaluation

Die Evaluation beginnt mit der Relation, die in Kapitel 3 und 4 als Beispiel für die Veranschaulichung der behandelten Grundlagen und Algorithmen dient. Die Relation auf der Menge $X = \{a, b, c, d, e\}$ mit den Relation $R = \{(a, b), (b, c), (c, d), (d, c)\}$

stellt einen sehr guten Testfall dar, da sie sowohl Suzumura-konsistent ist als auch mit e ein „freies“ Element enthält.

Basierend auf der theoretischen Analyse sollten sieben totale Ordnungserweiterungen generiert werden, die alle möglichen Positionierungen des „freien“ Elements e berücksichtigen, während die Äquivalenz zwischen den Elementen c und d erhalten bleibt. Diese sieben totalen Ordnungserweiterung werden, wie Abbildung 3 zeigt, wie erwartet von der Implementierung ermittelt und ausgegeben.

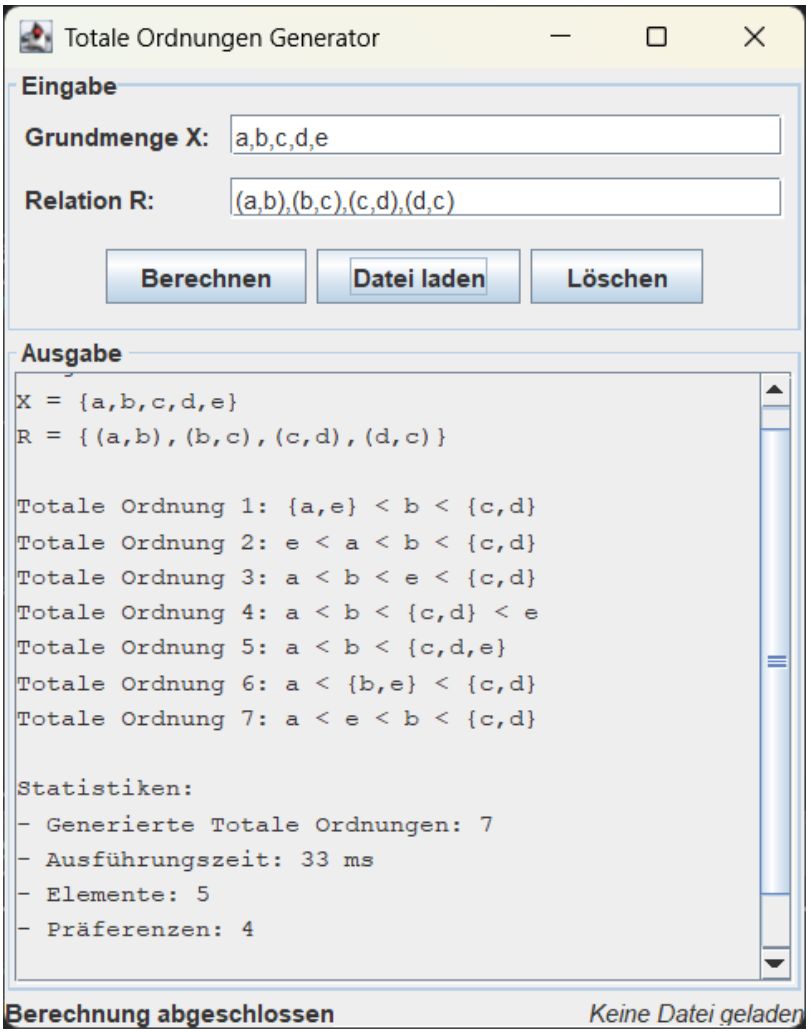

Abbildung 3: Ausgabe für Beispiel-Relation 1

Um ebenso darzustellen, dass die Implementierung auch nicht Suzumura-konsistente Relationen erkennt, wird der Beispiel-Relation die Präferenz (d, b) hinzugefügt, um so mit den Element b, c und d eine SCC zu bilden, die jedoch nicht vollkommen symmetrisch ist und somit einen PR-Zyklus bildet, der der Suzumura-

Konsistenz widerspricht. Somit lautet die Eingabe: Menge $X = \{a, b, c, d, e\}$ und Relation $R = \{(a, b), (b, c), (c, d), (d, c), (d, b)\}$.

Dabei lautet die Ausgabe „Fehler: Der asymmetrische Zyklus {b,c,d} verletzt die Suzumura-Konsistenz.“. Dies zeigt, dass der implementierte Algorithmus zum einen nicht Suzumura-konsistente Relationen erkennt und zum anderen eine aussagekräftige Fehlermeldung ausgibt, die die Information enthält, durch welchen PR-Zyklus die Suzumura-Konsistenz verletzt ist.

Für den Abschluss der qualitativen Betrachtung der Implementierung wird die Beispiel-Relation zum einen um das Element f und zum anderen um die Präferenz (e, f) ergänzt. Diese Relation wird gewählt, um zu zeigen, welche Auswirkungen es nach sich zieht, wenn man eine der beiden nicht vergleichbaren Teil-Relationen erweitert.

Während es in Beispiel 1 eine Teil-Relation mit drei Äquivalenzklassen - Elemente a, b, c und d - und eine Teil-Relation mit einer Äquivalenzklasse - Element e - gibt, wird die Anzahl der Äquivalenzklassen der zweiten Teil-Relation durch das Hinzufügen des Elements f auf zwei vergrößert.

Mithilfe der Delannoy-Zahlen lässt sich, wie bereits erwähnt, die Anzahl der totalen Ordnungserweiterungen bei zwei nicht vergleichbaren Teil-Relationen ermitteln. Während die Delannoy-Zahl(3,1) die in Beispiel 1 konstruierten sieben Erweiterungen bestätigt, ergeben sich nach der Delannoy-Zahl(3,2) bereits 25 totale Ordnungserweiterungen. Startet man die Implementierung mit der Eingabe Menge $X = \{a, b, c, d, e, f\}$ und Relation $R = \{(a, b), (b, c), (c, d), (d, c), (e, f)\}$, erhält man die erwarteten 25 totalen Ordnungserweiterungen.

Dies zeigt, dass der Algorithmus qualitativ korrekt arbeitet und sowohl alle möglichen totalen Ordnungserweiterungen konstruiert, als auch nicht Suzumura-konsistente Relationen erkennt.

6.2 Quantitative Evaluation

Bei der quantitativen Analyse werden nun Eingaben getestet werden, bei der eine sehr hohe Anzahl an totalen Ordnungserweiterungen erwartet wird. Dabei wird sowohl die Anzahl der generierten Ordnungserweiterungen mit der erwarteten Anzahl abgeglichen, als auch die Entwicklung der Laufzeit im Vergleich zur Anzahl der konstruierten Ordnungserweiterungen betrachtet.

Dafür wird jeweils eine Relation mit sieben, acht, neun und zehn Elementen in X, jedoch ohne eine Präferenz, als Eingabe betrachtet. Die Anzahl der möglichen linearen Ordnungserweiterungen einer Relation mit n Elementen und ohne Präferenzen kann dann durch die Anzahl der Permutationen, also durch die Fakultät der Anzahl der Elemente, angegebenen werden [14]. Dem entgegen gibt es durch die erlaubten Äquivalenzen zwischen den Elementen wesentlich mehr totale als lineare

Ordnungserweiterungen von Relationen ohne Präferenzen. Diese kann durch die Fubini-Zahlen angegeben werden [18].

Dementsprechend wird für eine Relation ohne Präferenzen und mit einer Element-Menge, die aus sieben Elementen besteht, die Konstruktion von 49.273 totalen Ordnungserweiterungen erwartet. Die entspricht auch der Ausgabe der Implementierung, die diese 47.293 totalen Ordnungserweiterungen nach 347 ms in einer txt-Datei ausgibt.

Fügt man der Element-Menge noch ein weiteres Element hinzu, gibt es bereits 545.835 mögliche totale Ordnungserweiterungen. Auch hier arbeitet die Implementierung korrekt und hat nach 3.930 ms den Vorgang abgeschlossen und alle Ordnungserweiterungen in eine txt-Datei geschrieben.

Ebenso die Testläufe mit neun und zehn Elementen geben die korrekte Anzahl totaler Ordnungserweiterungen aus und geben 7.087.261 totale Ordnungserweiterungen in 55.311 ms bzw. 102.247.563 totale Ordnungserweiterungen in 892.363 ms aus. Tabelle 1 stellt die Ergebnisse der vier Läufe übersichtlich dar und zeigt, dass die Laufzeit pro konstruierter Ordnungserweiterung bei sehr hohen Ausgaben leicht ansteigt, jedoch mit einer Laufzeit von weniger als 0,01 ms pro konstruierter totaler Ordnungserweiterung noch sehr effizient ist.

Anzahl Element	7	8	9	10
Anzahl Ordnungserweiterungen	49.273	545.835	7.087.261	102.247.563
Ausführungszeit in ms	347	3.930	55.311	892.363
Ausführungszeit je Ordnungserweiterung in ms	0,07	0,07	0,08	0,09

Tabelle 1: Anzahl totaler Ordnungserweiterungen und Ausführungszeiten

7 Fazit und Empfehlungen

Diese Arbeit hat das Ziel verfolgt, ein konstruktives Verfahren zur Erzeugung aller totalen Ordnungserweiterungen aus Suzumura-konsistenten Relationen zu entwickeln und zu implementieren. Nach der Analyse bestehender theoretischer Ansätze und der Entwicklung eines Verfahrens, für welches bewährte Algorithmen aus der Graphentheorie angepasst und miteinander kombiniert wurden, konnten die aufgezeigten Herausforderungen dieser Problemstellung erfolgreich bewältigt werden.

Zusammenfassung der wesentlichen Ergebnisse

Zunächst wurden die Grenzen bestehender Ansätze aufgezeigt. Sowohl Suzumuras als auch Andrikopoulos' Existenzbeweis erwiesen sich als für ein konstruktives

Verfahren ungeeignet, da sie auf dem Zornschen Lemma basieren, welches einen nicht–konstruktiven Existenzsatz darstellt. Das klassische topologische Sortieren ist algorithmisch effizient, scheitert aber an der Behandlung äquivalenter Elemente, die in Suzumura-konsistenten Relationen erlaubt sind.

Als Lösung wurde ein Verfahren entwickelt, das die Vorteile verschiedener behandelter Ansätze kombiniert und das Problem in zwei Phasen unterteilt: Zunächst wird die Eingabe-Relation durch eine angepasste Variante des Tarjan-Algorithmus in Äquivalenzklassen unterteilt und die Suzumura-Konsistenz überprüft. Anschließend wird ein erweiterter Kahn-Algorithmus angewendet, der nicht nur eine lineare, sondern alle totalen Ordnungserweiterungen durch die Berücksichtigung von Äquivalenzen und Betrachtung aller möglichen nicht-leeren Teilmengen in jeder Iteration konstruiert.

Die theoretische Analyse konnte belegen, dass das entwickelte Verfahren eine lineare Laufzeit von $O(n + m)$ pro generierter totaler Ordnungserweiterung aufweist, wobei n die Anzahl der Elemente und m die Anzahl der Präferenzen bezeichnet.

Bewertung der Implementierung

Die praktische Umsetzung des Algorithmus in Java unter Verwendung einer Layered Architecture zeigt die erfolgreiche Implementierung der theoretischen Konzepte. Zudem zeigt die Evaluation anhand verschiedener ausgewählter Testfälle sowohl die funktionale Korrektheit als auch die Effizienz des Verfahrens.

Besonders hervorzuheben ist die Robustheit des implementierten Ansatzes bei der Behandlung großer Ergebnismengen. Die quantitative Evaluation zeigt, dass selbst bei der Generierung von über 100 Millionen totalen Ordnungserweiterungen eine Laufzeit von weniger als 0,1 ms pro konstruierter totaler Ordnungserweiterung erreicht wurde. Dies wurde unter anderem durch die implementierte Streaming-Ausgabe ermöglicht, die den Speicherbedarf unabhängig von der Anzahl der Ergebnisse konstant hält.

Dennoch zeigte die Evaluation auch ein Problem der aktuellen Implementierung. Bei der Ausgabe sehr großer Ergebnismengen über die GUI tritt ab etwa 6 Millionen generierten Ordnungserweiterungen ein Heap-Overflow auf. Interessant ist dabei, dass dieses Problem nur die Ausgabe in der GUI betrifft, während die Ausgabe in txt-Dateien auch bei wesentlich größeren Ergebnismengen problemlos funktioniert. Für produktive Anwendungen mit sehr großen Ergebnismengen ist daher die Verwendung der Datei-basierten Ein- und Ausgabe zu empfehlen.

Anwendungsfelder und praktische Relevanz

Die entwickelte Konstruktionsmethode ermöglicht vielfältige Anwendungsmöglichkeiten in verschiedenen Bereichen der Informatik und Wirtschaftswissenschaf-

ten. Bei Datenbanken können totale Ordnungserweiterungen zur Optimierung von Sortier- und Indexierungsoperationen eingesetzt werden, insbesondere wenn nur partielle Ordnungsinformationen verfügbar sind. Dies ist bspw. bei der Behandlung von Datensätzen mit unvollständigen Informationen oder bei der Integration verschiedener Datenquellen mit unterschiedlichen Ordnungskriterien relevant [13].

Zudem ist das Verfahren für das Anwendungsfeld der Produktions- und Arbeitsablaufplanung interessant. In vielen industriellen Prozessen existieren partielle Ordnungsbeziehungen zwischen Arbeitsschritten, die durch technische Abhängigkeiten, Ressourcenverfügbarkeit oder logistische Einschränkungen entstehen. Die Konstruktion aller möglichen totalen Ordnungserweiterungen ermöglicht es Planern, sämtliche zulässigen Produktionsreihenfolgen zu identifizieren und daraus die unter gegebenen Kriterien optimale auszuwählen. Dies ist insbesondere in flexiblen Fertigungssystemen von Bedeutung, in denen die Möglichkeit zur dynamischen Anpassung der Produktionsreihenfolge einen entscheidenden Wettbewerbsvorteil darstellt [21].

Weiterentwicklung der Software

Die aktuelle Implementierung bietet eine solide Grundlage für verschiedene Erweiterungen und Verbesserungen. Im ersten Schritt sollte die identifizierte Heap-Overflow-Problematik in der Beutzeroberfläche behoben werden.

Aus funktionaler Sicht würde die Software von der Integration zusätzlicher Ausgabeformate profitieren. Neben der aktuellen txt- und GUI-Ausgabe wären strukturierte Formate wie JSON oder XML für die Weiterverarbeitung in anderen Systemen hilfreich. Ebenso könnte eine grafische Visualisierung der Eingabe-Relationen und der generierten totalen Ordnungserweiterungen das Verständnis des Sachverhalts erleichtern.

Die Implementierung weiterer Konsistenzprüfungen über die Suzumura-Konsistenz hinaus würde die Anwendbarkeit der Software erweitern. So würde die Integration von Tests auf bspw. Δ-Konsistenz und Transitivität ermöglichen, das Verfahren auch für ähnliche Problemstellungen nutzbar zu machen.

Aus Performance-Sicht wäre die Implementierung einer Parallelisierung des Algorithmus interessant, da die Generierung der totalen Ordnungserweiterungen im Kahn-Algorithmus unabhängig erfolgt und bspw. eine parallele Bearbeitung von Schleifen-Durchläufen möglich wäre, was insbesondere bei sehr großen Ergebnismengen zu Laufzeitverbesserungen führen könnte.

Zudem zeigt die erfolgreiche Implementierung des Verfahrens, dass sich auch andere Algorithmen für die Konstruktion linearer Erweiterungen für die Konstruktion totaler Ordnungserweiterungen anpassen lassen könnten. Interessant ist bspw. der

in Kapitel 1 schon erwähnte Pruesse-Ruskey-Algorithmus, der für die effiziente Erzeugung aller linearen Erweiterungen einer Relation entwickelt wurde [22]

Ausblick und wissenschaftlicher Beitrag

Diese Arbeit trägt dazu bei, den Übergang zwischen theoretischen Existenzbeweisen zu einem praktischen Konstruktionsverfahren für totale Ordnungserweiterungen Suzumura-konsistenter Relationen zu ermöglichen. Durch die erfolgreiche Kombination und Anpassung bekannter Algorithmen wurde ein Verfahren entwickelt, das sowohl theoretisch fundiert als auch praktisch nutzbar ist.

Der entwickelte Ansatz zeigt, wie theoretische Erkenntnisse der Präferenz- und der Graphentheorie erfolgreich kombiniert werden können, was als Modell für die Bearbeitung ähnlicher Problemstellungen dienen könnte.

Zusammenfassend verdeutlicht die Arbeit, dass die Konstruktion totaler Ordnungserweiterungen Suzumura-konsistenter Relationen nicht nur theoretisch möglich, sondern auch praktisch effizient realisierbar ist. So stellt die entwickelte Implementierung ein funktionsfähiges Werkzeug dar, das sowohl für Forschungszwecke als auch für praktische Anwendungen genutzt werden kann und damit einen wertvollen Beitrag zur Präferenztheorie leistet.

Literatur

[1] Athanasios Andrikopoulos. Szpilrajn-type theorems in economics. 2009.

[2] Jérémy Barbay, Francisco Claude, and Gonzalo Navarro. Compact binary relation representations with rich functionality. *Information and Computation*, 232:19–37, 2013.

[3] Walter Bossert, Susumu Cato, and Kohei Kamaga. Independent, neutral, and monotonic collective choice: the role of Suzumura consistency. *Social Choice and Welfare*, 61(4):835–852, 2023.

[4] Walter Bossert, Susumu Cato, and Kohei Kamaga. Smallest quasi-transitive extensions. *Journal of Mathematical Economics*, 112:102983, 2024.

[5] Walter Bossert and Kotaro Suzumura. *Consistency, choice, and rationality*. Harvard University Press, 2010.

[6] Graham Brightwell and Peter Winkler. Counting linear extensions is# p-complete. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 175–181, 1991.

[7] Susumu Cato. Szpilrajn, Arrow and Suzumura: concise proofs of extension theorems and an extension. *Metroeconomica*, 63(2):235–249, 2012.

[8] Kwang-Wu Chen. Delannoy numbers and preferential arrangements. *Mathematics*, 7(3):238, 2019.

[9] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[10] Brian A Davey and Hilary A Priestley. *Introduction to lattices and order*. Cambridge university press, 2002.

[11] Peter C Fishburn. *The theory of social choice*. Princeton University Press, 1973.

[12] Cristian Frăsinaru and Emanuel Florentin Olariu. Graph4j–a computationally efficient java library for graph algorithms. *arXiv preprint arXiv:2308.09920*, 2023.

[13] Goetz Graefe. Implementing sorting in database systems. *ACM Computing Surveys (CSUR)*, 38(3):10–es, 2006.

[14] David Guichard. An introduction to combinatorics and graph theory. *Whitman College-Creative Commons*, 2017.

[15] Bengt Hansson. Choice structures and preference relations. *Synthese*, 18(4):443–458, 1968.

[16] Arthur B Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562, 1962.

[17] Ralph L Keeney and Howard Raiffa. *Decisions with multiple objectives: preferences and value trade-offs*. Cambridge university press, 1993.

[18] OEIS Foundation Inc. Entry A000670 in the on-line encyclopedia of integer sequences. `https://oeis.org/A000670`, 2025. Abgerufen am 15.08.2025.

[19] Oracle. The list interface. `https://docs.oracle.com/javase/tutorial/collections/interfaces/list.html`, 2014. Abgerufen am 15.08.2025.

[20] Oracle. The set interface. `https://docs.oracle.com/javase/tutorial/collections/interfaces/set.html`, 2014. Abgerufen am 15.08.2025.

[21] Michael Pinedo and Khosrow Hadavi. Scheduling: theory, algorithms and systems development. In *Operations research proceedings 1991: Papers of the 20th annual meeting/vorträge der 20. Jahrestagung*, pages 35–42. Springer, 1992.

[22] Gara Pruesse and Frank Ruskey. Generating linear extensions fast. *SIAM Journal on Computing*, 23(2):373–386, 1994.

[23] Kotaro Suzumura. Remarks on the theory of collective choice. *Economica*, pages 381–390, 1976.

[24] Edward Szpilrajn. Sur l'extension de l'ordre partiel. *Fundamenta mathematicae*, 1(16):386–389, 1930.

[25] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[26] Zhenan Tu. Research on the application of layered architecture in computer software development. *J. Comput. Electron. Inf. Manag*, 11(3):34–38, 2023.

[27] Peter Zeller and Lukas Stevens. Order Extension and Szpilrajn's Theorem. 2024.

[28] Max Zorn. A remark on method in transfinite algebra. *Bulletin of the American Mathematical Society*, 41(10):667–670, 1935.