Semantische Indexierung mit expliziten Wissensressourcen

Florian Fink

Inauguraldissertation
zur Erlangung des Doktorgrades der Philosophie
an der Ludwig-Maximilians-Universität

vorgelegt von
Florian Fink
aus München
2018

Erstgutachter: Prof. Dr. Klaus U. Schulz
Zweitgutachter: PD Dr. Stefan Langer
Tag der mündlichen Prüfung: 09. Juli 2018

Im Gedenken an
Elisabeth Gebhart-Fink
* 15. Januar 1955
† 11. September 2016

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Zusammenfassung

Die *semantische Indexierung mit expliziten Wissensressourcen* behandelt die Indexierung von Dokumentkollektionen mit Hilfe explizit ausgezeichneten Wissens. Hierbei soll das vorhandene Wissen mit in den Index aufgenommen werden und so für Suchanfragen bereitstehen.

Im ersten Teil dieser Arbeit werden verschiedene Arten wie explizites Wissen formalisiert wird im Hinblick auf eine semantische Indexierung untersucht und eine Reihe linguistischer Aspekte und deren Einfluß auf die Indexierung dargestellt. Auch werden algorithmische Verfeinerungen diskutiert, mit denen große Lexika für diese Form der Indexierung benutzt werden können.

Im zentralen Teil dieser Arbeit wird die semantische Indexierung vom Aufbau des Index über dessen Erzeugung bis hin zu den unterschiedlichen Suchanfragemöglichkeiten auf der Basis zweier unterschiedlicher Wissensressourcen erläutert. Ebenso werden verschiedene Aspekte der Implementierung einer solchen Indexierung anhand eines, parallel zu dieser Arbeit erzeugten Programms, erläutert.

Im letzten Teil dieser Arbeit werden verschiedene Möglichkeiten einer fehlertoleranten Indexierung sowie die Behandlung und Auflösung von Ambiguitäten untersucht.

Auch wird eine, parallel zu dieser Arbeit erzeugte Implementierung einer semantischen Indexierung mit expliziten Wissensressourcen, dargestellt. Dabei wird sowohl auf die allgemeine Verwendung der Anwenung als auch auf die Installation eingegangen.

Vorwort

Die Erzeugung eines Index im Information Retrieval dient vor allem dem schnellen Auffinden von relevanten Dokumenten in großen Dokumentenkollektionen. Dabei können die Dokumentenkollektionen wissenschaftliche Arbeiten, Bücher, Webseiten und vieles mehr umfassen. Durch die Hinzunahme von explizit ausgezeichnetem Weltwissen in den Indexierungsprozess können die Möglichkeiten der Suchanfragen erweitert werden sowie die Relevanz der Suchergebnisse verbessert werden.

Die Digitalisierung unseres täglichen Lebens schreitet unaufhaltsam voran. Sie umfasst dabei viele unterschiedliche Bereiche unseres täglichen Lebens – darunter vor allem auch Textdokumente jeder Art. Bibliotheken bieten heutzutage ihre Dokumente, seien es nun Bücher, Journale oder Fachartikel, zusätzlich auch digital über das Netz an. Neuerscheinungen von Büchern und Zeitungen werden verstärkt auch digital veröffentlicht. Verschiedene Artikel werden sogar nur noch in digitaler Form verfügbar gemacht. Dies führt zu einer stetigen Vergrößerung des textuellen Datenbestandes. Während die Digitalisierung auf der einen Seite die Verfügbarkeit von Informationen verbessert, führt die wachsende Informationsmenge auch zu einer immer schwieriger werdenden Auffindbarkeit von spezifischen relevanten Informationen.

Das Fachgebiet *Information Retrieval (IR)* beschäftigt sich mit genau dieser Problemstellung. Es behandelt die computergestützten Informationsgewinnung aus einem digitalen Datenbestand. Im Allgemeinen kann der betrachtete Datenbestand eine Vielzahl von unterschiedlichen Daten enthalten: Von einfachen Textdokumenten über Bilder und Sprachdaten hin zu Audio- und Videodateien. Das sogenannte *Dokumentenretrieval* oder *Document Retrieval* beschäftigt sich dagegen mit dem Auffinden von Textdokumenten in einer Text- bzw. Dokumentenkollektion. Für Suchanfragen nach einem oder mehreren Schlagwörtern sollen die relevanten Dokumente, das heißt diejenigen Dokumente, welche die angefragten Informationen enthalten, in der Dokumentenkollektion gefunden und zurückgeliefert werden.

Hierzu werden die Dokumente der Kollektion nach bestimmten Schlüsselwörtern durchsucht und die Ergebnisse in einem zentralen *Index* abgelegt. Mit Hilfe dieses Index können dann Dokumente für Suchanfragen schnell aufgefunden wer-

den, ohne jedes Mal die gesamte Dokumentenkollektion durchsuchen zu müssen. Dieser Index entspricht dabei dem aus Büchern bekannten Index. Für bestimmte Schlüsselwörter im Index kann ein interessierter Leser dann die relevanten Seiten im Buch nachschlagen.

Die übliche Indexierung von Dokumenten im Rahmen des Information Retrievals verwendet zur internen Repräsentation der Dokumente nur deren Einzelwörter [66]. Dabei werden üblicherweise bestimmte, häufig auftretende Wörter – sogenannte *Stoppwörter* – eliminiert, da sie für sich alleine gesehen nur sehr wenig bis gar keine relevante Information tragen. Oftmals wird durch weitere wortbasierte und weitere linguistische Normalisierungen wie *Stemming* und *Lemmatisierung* die Menge der Wörter in der internen Dokumentenrepräsentation weiter verallgemeinert.

Diese als *Bag-of-words* bezeichnete Dokumentenrepräsentation betrachtet Dokumente als einfache Ansammlung von Einzelwörtern. Es ist über die Dokumente nur bekannt, welche Einzelwörter mit welcher Häufigkeit in einem Dokument vorgefunden wurden. Die interne Repräsentation beachtet weder die Reihenfolge der Wörter im Ursprungsdokument noch komplexere, aus mehreren Wörtern zusammengesetzte sprachliche Ausdrücke.

Zur Erzeugung des Index muss die interne Darstellung der Dokumente als Ansammlung von Schlüsselwörtern so umgekehrt werden, dass die Schlüsselwörter den Dokumenten, in denen sie vorkommen, zugeordnet sind. Erst diese als *invertierte Listen* bezeichnete Datenstrukturen ermöglichen dann eine effiziente Suche nach Schlüsselwörtern des Index und ermöglichen es einfach auf die Dokumente zugreifen zu können, in denen angefragte Suchbegriffe vorkommen.

Das *Vektorraummodell (VRM)* [48] dagegen repräsentiert die Dokumente als Vektoren von Vorkommenshäufigkeiten, wobei jedem Schlüsselwort des Index eine eigenen Dimensionalität zugeordnet wird. Der Index beim VRM besteht aus einer Dokument-Term Matrix, die alle Dokumentenvektoren des Index umfasst. Suchanfragen werden in solchen System genau wie Dokumente behandelt. Sie werden auf dieselbe Weise wie die Dokumente in eine interne Repräsentation überführt. Zur Berechnung der Ergebnismenge von Suchanfragen werden die internen Repräsentationen der Dokumente in der Kollektion mit der Repräsentation der Suchanfrage verglichen und die ähnlichsten Dokumente ausgewählt. Hierzu werden verschiedene spezialisierte Abstandsmaße verwendet, die es ermöglichen, die Ähnlichkeit von Dokumenten zueinander zu bestimmen.

Um die interne Dokumentenrepräsentation beim Dokumentenretrieval zu verbessern und somit auch die Retrievalergebnisse, versucht nun die *semantische Indexierung* verschiedene semantische Informationen in die interne Dokumentenrepräsentation der Indexierungsverfahren mit einfließen zu lassen. Bisher werden semantische Beziehungen zwischen Begriffen vor allem durch Synonymlisten auf

der Basis von Einzelwörtern [29, S. 131] explizit in die Dokumentenrepräsentation eingebracht. So sollen die Dokumentenrepräsentation und die Suchanfragen unabhängig von sprachlicher Variation gemacht werden. Die Ergebnisse von Suchanfragen können so auch Dokumente mit synonymen Wörtern und Formulierungen liefern. Andere Verfahren, wie die *latent semantische Indexierung* [13], zielen darauf ab, eingeschränkt vage Wortähnlichkeiten zwischen Schlüsselwörtern implizit über die Einbeziehung von kombinierten Auftretenswahrscheinlichkeiten von Wörtern in die Dokumentenrepräsentation einzubinden. So kann eine statistische Form von Synonymen und Wortähnlichkeiten implizit aus den Dokumenten selbst hergeleitet werden, ohne auf statische und unter Umständen unvollständige Synonymlisten zurückgreifen zu müssen.

Im Gegensatz zu den gerade dargestellten Ansätzen des Dokumentenretrievals und der latenten semantischen Indexierung, wird in dieser Arbeit die *semantische Indexierung mit expliziten Wissensressourcen* behandelt. Diese Form der Indexierung erzeugt einen durchsuchbaren Index, der aus den in den expliziten Wissensressourcen ausgezeichneten Beziehungen aufgebaut wird. Hierzu werden die in den verwendeten Wissensressourcen ausgezeichneten Beziehungen zwischen eindeutigen *Konzepten* zur Indexierung von Dokumenten herangezogen. Die verwendeten Wissensressourcen modellieren die Welt über eine Menge eindeutiger Konzepte und deren Verbindungen untereinander. Genau diese Konzepte und deren Verbindungen untereinander werden bei der semantischen Indexierung verwendet, um die in den Wissensressourcen ausgezeichneten, logisch-semantischen Verbindungen in den Index zu übernehmen.

Wann immer in der expliziten Wissensressource eine Verbindung $A \rightarrow B$ zwischen zwei Konzepten A und B ausgezeichnet ist, wird bei der semantischen Indexierung ein Dokument d, für welches ein Indexeintrag für das Konzept A erzeugt werden soll, auch ein entsprechender Indexeintrag für das Konzept B erzeugt. So können Suchanfragen nach sowohl dem Konzept A als auch dem Konzept B das indexierte Dokument d zurück liefern.

Konkret bedeutet dies, dass ein Dokument, welches mit dem Konzept *Peking* indexiert wurde, auch mit dem Konzept *China* indexiert wird, sofern die verwendete Wissensressource ein Verbindung zwischen den Konzepten *Peking* $\rightarrow$ *China* auszeichnet. Suchanfragen nach dem Konzept *China* liefern so auch dieses, mit *Peking* indexierte Dokument zurück. Die in der Wissensressource ausgezeichnete Beziehung wird bei der semantischen Indexierung mit berücksichtigt und ist explizit im semantischen Index hinterlegt.

Bei dieser Form der Indexierung können darüber hinaus nicht nur die Beziehungen zwischen den Konzepten, sondern auch die genaue Art ihrer Beziehung mit berücksichtigt werden. Je nach verwendeten Wissensressourcen und den genauen, in ihnen ausgezeichneten logisch-semantischen Beziehungen kann, bei der

semantischen Indexierung dieses kodierte Wissen mit in den Index übernommen werden. Es steht somit auch für Suchanfragen zur Verfügung.

In dieser Arbeit wird daher auch der Frage nachgegangen werden, in wie weit *Weltwissen*, welches durch verschiedene Thesauri, Ontologien und anderen Wissensbasen *explizit* ausgezeichnet ist, bei der semantischen Indexierung eingebunden werden kann. Es soll zunächst dargestellt werden, wie das Wissen aus verschiedenen expliziten Wissensressourcen zur Erfassung von Themen in Dokumentenkollektionen eingesetzt werden kann. Hierzu müssen auch unterschiedliche Formen, in denen Weltwissen repräsentiert wird, analysiert werden. Neben der Fragestellung der Repräsentation von Wissensressourcen, müssen darüber hinaus auch die in diesen Ressourcen auftretenden Themen in den natürlichsprachlichen Dokumenten der Kollektion identifiziert werden, wobei auch verschiedene linguistische Problemstellungen behandelt werden müssen.

Das vorhandene Wissen in den expliziten Wissensressourcen der semantischen Indexierung kann nicht nur bei der Indexierung und Klassifizierung von Dokumenten verwendet werden. Es ist auch naheliegend dieses Wissen bei Suchanfragen und somit auch bei der Dokumentensuche mit einzubeziehen. Daher werden in dieser Arbeit auch Möglichkeiten aufgezeigt, inwieweit das Weltwissen zur Erweiterung von Suchanfragen herangezogen werden kann, und welche Möglichkeiten eine solche erweiterte Suche bietet.

Wie bereits erwähnt, kann das hier behandelte Weltwissen aus verschiedenen Quellen stammen. Heutzutage stehen unterschiedliche Ressourcen zur Verfügung, die allesamt versuchen, das menschliche Wissen auf eine formalisierte Weise abzubilden. Sie werden im weiteren Verlauf dieser Arbeit gleichermaßen als *Wissensbasis*, *Wissensquelle* oder *Wissensressource* bezeichnet. Allen hier behandelten Wissensbasen ist gemein, dass sie — stark vereinfacht ausgedrückt — eine Menge von *Konzepten* zueinander in Beziehung setzen, wobei diese Beziehungen der Konzepte untereinander die logisch-semantische Repräsentation der Welt oder eines kleineren, spezialisierten Teilgebietes modellieren.

Konzepte in diesen Wissensressourcen sind immer eindeutig. Sie referenzieren immer reale oder abstrakte Objekte einer Domäne und sind in eine mehr oder weniger stark formalisierte Hierarchie eingebettet. Je nach verwendeter Wissensressource können die Konzepte benannte Entitäten, Personen, Orte, Organisationen, abstrakte Ideen und vieles mehr umfassen. Die Menge der Konzepte einer Wissensressource bildet dabei die Menge aller indexierbarer Objekte. Diese Menge ist aus offensichtlichen Gründen zugleich auch die Menge aller abfragbarer Objekte.

Alle Konzepte der Wissensressource müssen immer durch globale, eindeutige Bezeichner, sogenannte *Unique Resource Identifier (URIs)*, festgeschrieben sein. Damit sie auch für Menschen einfacher zu handhaben sind, können die Konzepte auch mit möglichst eindeutigen natürlichsprachlichen Vorzugsnamen benannt werden,

wobei bei dieser Art der Benennung neben der Eindeutigkeit keine weiteren Forderungen an die Benennung der Konzepte gestellt wird[1].

Die grundlegende Annahme dieser Arbeit ist, dass in sinnvollen natürlichsprachlichen Dokumenten verschiedene Konzepte einer Wissensbasis *direkt* im Text über bestimmte Schlüsselwörter kodiert sind. Neben den direkt kodierten Konzepten können die Dokumente darüber hinaus weitere Themen enthalten, die nicht explizit ausgezeichnet sind, sondern implizit über verschiedene logisch-semantische Beziehungen zu den direkt ausgezeichneten Konzepten.

Es können somit neben den direkten Konzepten in Dokumenten noch weitere Konzepte zur thematischen Einordnung identifiziert werden. Diese Konzepte sind eben nicht direkt in den Dokumenten durch verschiedene Schlüsselwörter realisiert, sondern müssen *indirekt* über die logisch-semantischen Verbindungen der Konzepte in den Wissensressourcen identifiziert werden.

Die Menge aller Konzepte, die entweder direkt über Schlüsselwörter oder indirekt über die Verbindungen der Konzepte untereinander in einem Dokument identifiziert werden können, bilden dann die thematische Einordnung eines Dokuments, welche die Grundlage der semantischen Indexierung von Dokumenten bildet. Hierbei definieren die expliziten Wissensressourcen über die Verbindungen ihrer Konzepte untereinander — neben den explizit in den Dokumenten ausgezeichneten Konzepten — weitere implizite thematische Einordnungen für die Dokumente.

Die direkten Konzepte sind dabei oftmals nicht nur durch einfache Schlüsselwörter in den Dokumenten kodiert. Vielmehr sind die Konzepte durch verschiedene sprachliche Ausdrücke repräsentiert und müssen mit geeigneten Verfahren in den Dokumenten identifiziert werden. Es muss also bei dieser Form der Indexierung möglich sein, die Vielzahl von sprachlichen Ausdrücken auf die tatsächlichen Konzepte in den Dokumenten abzubilden. Dabei müssen neben einfachen Schlüsselwörtern auch komplexere Mehrwortverbindungen betrachtet werden. Darüber hinaus muss auch die Problematik ambiger sprachlicher Ausdrücke berücksichtigt werden, die eindeutige Zuordnungen sprachlicher Ausrücke auf Konzepte erschweren.

Um bei der semantischen Indexierung mit expliziten Wissensressourcen die Konzepte in Dokumenten identifizieren zu können, enthalten die verwendeten Wissensressourcen neben Konzepten und deren Verbindungen auch ein Lexikon, welches verschiedene natürlichsprachliche Ausdrücke eindeutig auf die Konzepte der

[1] Ein Beispiel für eine solche natürlichsprachliche Benennung ist die Namenskonvention der Wikipedia. Hier werden ambige Ausdrücke einfach durch Hinzufügen von weiteren Bezeichnern unterschieden. So wird z.B. zwischen *Relation*, *Relation (Mathematik)* und *Relation (Datenbanken)* unterschieden, um gleich benannte Konzepte aus verschiedenen Gebieten unterscheiden zu können [65].

Wissensressource abbildet. Aus dem oben genannten Grund kann das Lexikon nicht nur Einzelworte enthalten, sondern auch komplexe Mehrwortverbindungen.

Die Identifizierung der Konzepte in den Dokumenten stellt den eigentlichen Indexierungsprozess dar. Konzepte, die bei der Indexierung in den Dokumenten vorgefundenen werden und deren Beziehungen untereinander, wie sie durch die Wissensbasis vorgegeben sind, können dann zu einer thematischen Einordnung der Dokumente herangezogen werden, wobei die Ergebnisse des Indexierungsprozesses — die in den Dokumenten identifizierten Konzepte — in einer Datenbank oder einer anderen geeigneten Struktur für die spätere Suche vorgehalten werden.

Durch den Aufbau spezieller computerlinguistischer Datenstrukturen aus den verwendeten Wissensressourcen kann bei der Indexierung auf das bekannte Vorgehen aus dem Information Retrieval verzichtet werden, aus der internen Dokumentenrepräsentation erst mit verschiedenen hochspezialisierten Algorithmen die bereits erwähnten invertierten Listen zu berechnen, die das eigentliche Retrieval von Dokumenten ermöglichen [66, S. 109ff]. Diese computerlinguistischen Datenstrukturen ermöglichen unter anderem einen direkten iterativen Aufbau des Index aus den Dokumenten, die Verwendung von Lexikoneinträgen beliebiger Länge und auch die einfache Einbeziehung von Konzeptbeziehungen aus der Wissensressource.

Die semantische Indexierung geht dabei einfach dokumentenweise vor. Ein als Trie gespeichertes Lexikon ist an seinen Finalzuständen direkt mit den Konzepten der Wissensressource verbunden, welche ihrerseits wiederum direkt mit ihren verbundenen Konzepten verbunden sind. Die Dokumente werden buchstabenweise durchlaufen. Parallel dazu werden die Zustände des Tries durchlaufen. Sobald der tiefste erreichbare Finalzustand erreicht wurde, kann ein entsprechender Textbeleg für das Konzept am Finalzustand des Tries indexiert werden. Da die Konzepte alle ihre Verbindungen kennen, können auch sofort entsprechenden Indexeinträge für die verbundenen Konzepte durchgeführt werden. Der Index besteht dabei aus einer Menge von Dateien, wobei jede Datei den Index für genau ein Konzept der Wissensressource repräsentiert und über dessen URI eindeutig identifiziert werden kann. Suchanfragen nach Konzepten können daher direkt die entsprechenden Indexdateien einlesen, in denen alle Textbelege zu angefragten Konzepten enthalten sind.

Grundsätzlich kann die semantische Indexierung mit expliziten Wissensressourcen auf unterschiedlichen textuellen Dokumenten angewendet werden — insbesondere auch auf allen Formen moderner Textdokumente wie Webseiten, Büchern oder Geschäftsdokumenten. Diese Dokumente bilden den Großteil der zu indexierenden Dokumente und können mit dem einfachen, in dieser Arbeit dargestellten, Basisverfahren zur semantischen Indexierung behandelt werden.

Neben modernen Dokumenten kann die semantische Indexierung auch auf wei-

tere, spezifischere Typen von Dokumenten angewendet werden. Darunter auch auf Texte mit Fehlern[2] oder auf historische Texte mit historischen Wortformen und -schreibungen. Diese spezielleren Typen von Dokumenten müssen bei der semantischen Indexierung gesondert behandelt werden, wobei der resultierende semantische Index ohne weitere Besonderheiten abgefragt und durchsucht werden kann. Um die Konzepte der Wissensressourcen in solchen fehlerbehafteten Dokumenten sicher identifizieren zu können, müssen bei der Indexierung spezielle Techniken verwendet werden, die approximative Suchen auf den Wörterbüchern ermöglichen und auch mit eventuell auftretenden Ambiguitäten umgehen können.

Die semantische Indexierung mit expliziten Wissensressourcen baut einen durchsuchbaren semantischen Index auf. Der Index kann dabei direkt mit den Wissensressourcen inkrementell aus den Eingabedokumenten aufgebaut werden (siehe oben), ohne intermediäre Zwischenformate zum Indexaufbau verwenden zu müssen.

Durch die direkte Anreicherung des Index mit dem semantischen Weltwissen der verwendeten Wissensressourcen während der Indexierung werden entsprechende thematische Abhängigkeiten und logisch-semantische Verbindungen der Konzepte in den Dokumenten explizit im Index gespeichert. Diese Verbindungen stehen dann für thematische Suchanfragen direkt zur Verfügung um unter anderem Über- und Unterbegriffe, thematische Felder und verschiedene logische Verbindungen von Begriffen in Suchanfragen mit zu berücksichtigen. Ebenso können die in den Wissensressourcen kodierten Verbindungen auch bei Suchanfragen herangezogen werden. Dies ermöglicht gezieltere Suchanfragen, welche die Verbindungen der Konzepte in den Wissensressourcen — und damit das in diesen abgespeicherte Wissen — ausnutzen können. So kann die Relevanz der zurückgelieferten Suchtreffer verbessert werden und gezielte Suchanfragen auf dem Index ermöglicht.

In dieser Arbeit werden die benötigten Datenstrukturen zur Speicherung der Konzepte und ihrer natürlichsprachlichen Bezeichner dargestellt, die zur semantischen Indexierung mit expliziten Wissensressourcen benötigt werden. Ebenso wird das auf den dargestellten Datenstrukturen basierende Indexierungsverfahren, welches den Index aus den Eingabedokumenten der Dokumentenkollektion aufbaut, sowie die Struktur des resultierenden Index dargestellt. Ausgehend von zwei unterschiedlichen realen Wissensressourcen werden sowohl einfache thematische Suche nach Schlüsselwörtern als auch Suchanfragen auf den benannten Relationen von Ontologien dargestellt. Abschließend werden Erweiterungen der se-

[2] Hierunter Fallen insbesondere auch Dokumente, die mit verschiedenen Erkennungsverfahren, wie *Optical Character Recognition (OCR)* oder *Speech Recognition (SR)* automatisch aus analogen Quellen erzeugt wurden.

mantischen Indexierung diskutiert, um einerseits fehlerbehaftete und historische Dokumente behandeln zu können und andererseits um Ambiguitäten bei der Indexierung auflösen zu können.

Die fünf Kernbeiträge dieser Arbeit lassen sich wie folgt zusammenfassen:

1. Darstellung minimierter Automaten zur Speicherung großer Lexika. Es werden die minimierten Automaten dargestellt, die zur Identifizierung von Konzepten in den Dokumenten verwendet werden und somit einen zentralen Bestandteil der semantischen Indexierung darstellen.

2. Darstellung der semantischen Indexierung mit expliziten Wissensressourcen. Aufbauend auf den Arbeiten in [19] wird die semantische Indexierung sowohl auf EFGT-Netzen als auch auf allgemeineren Ontologien dargestellt.

3. Darstellung der fehlertoleranten bzw. unscharfen Suche auf dem Lexikon der semantischen Indexierung. Es wird das Verfahren dargestellt, welches eine approximative Suche auf dem Automaten ermöglicht und so auch eine semantische Indexierung auf fehlerbehafteten oder historischen Dokumenten ermöglicht.

4. Darstellung der Disambiguierung ambiger Ausdrücke in den Dokumenten. Die semantische Indexierung ermöglicht eine kontextsensitive Disambiguierung ambiger natürlichsprachlicher Ausdrücke in den Dokumenten, welche erst durch die Verarbeitung der Dokumente mit expliziten Wissensressourcen ermöglicht wird.

5. Darstellung verschiedener Implementierungsaspekte der semantischen Indexierung mit expliziten Wissensressourcen. Es werden verschiedene Aspekte der Implementierung der semantischen Indexierung auf Basis der parallel zu dieser Arbeit entwickelten Anwendung zur semantischen Indexierung mit expliziten Wissensressourcen dargestellt.

Kapitel 1 behandelt die formalen Grundlagen dieser Arbeit. Es wird ein ein Überblick über alle nötigen Formalismen zur Darstellung von Relationen, formalen Sprachen und Automaten und zur Berechnung von Wortabstands- und ähnlichkeitsmaßen behandelt. Das Kapitel schließt mit der formalen Darstellung der in dieser Arbeit verwendeten, expliziten Wissensressourcen ab. Diese bilden die Grundlage für die weiteren Betrachtungen der semantischen Indexierung mit expliziten Wissensressourcen.

Kapitel 2 beleuchtet verschiedene linguistische Aspekte, die vor allem bei der Identifizierung der Konzepte in natürlichsprachlichen Dokumenten eine Rolle spielen. Dabei wird deren Einfluss auf die Dokumentenindexierung dargestellt. Neben

der Behandlung von Flexion und Derivation werden insbesondere auch komplexe Mehrwortverbindungen, historische Schreibvarianten und orthographische Fehler sowie Ambiguitäten behandelt.

In Kapitel 3 werden bekannte Verfahren aus der Computerlinguistik zur effizienten Verarbeitung der Wissensressourcen und der Dokumentenkollektion behandelt, die gerade im Information Retrieval nicht sehr weit verbreitet sind und somit selten in diesem Gebiet eingesetzt werden. Insbesondere wird ein Algorithmus dargestellt, der Wörterbücher effizient in einer Automatenstruktur abspeichern kann.

Kapitel 4 stellt die unterschiedlichen Formen vor, in denen verschiedene reale Wissensressourcen repräsentiert werden. Es werden sowohl stark hierarchische *tiefe* Strukturen behandelt, als auch weniger hierarchische *flache* Strukturen. Einzelne kurze Beispiele für die Wissensressourcen in diesem Kapitel werden in den weiteren Kapiteln zur Verdeutlichung eingesetzt.

In Kapitel 5 werden nun die bis hierher dargestellten Punkte kombiniert und es wird ein Verfahren zur semantischen Indexierung von Dokumentenkollektionen vorgestellt. Dieses Verfahren wird dann auf zwei unterschiedliche reale Wissensbasen angewendet und auf verschiedene Aspekte hin durchleuchtet. Dabei sollen vor allem auch die neuen Abfragemöglichkeiten dargestellt werden, die die Verwendung einer Wissensbasis zur Textindexierung ermöglicht.

Kapitel 6 stellt eine Erweiterung des bestehenden Systems vor, welches es ermöglicht Dokumente mit historischen Schreibvarianten oder orthographischen Fehlern zu behandeln. Dieses Verfahren ist gleichzeitig auch in der Lage, bestimmte Fehler in der Abfrage des Dokumentenindex zu behandeln.

Im letzten regulären Kapitel 7 werden abschließend Verfahren zur Behandlung und Auflösung von Ambiguitäten diskutiert. Diese Verfahren ermöglichen eine kontextbezogene Disambiguierung ambiger Begrifflichkeiten sowohl in den Wissensressourcen als auch in den Indexdokumenten.

Teil dieser Dissertation ist auch eine Anwendungssoftware mit den Namen *semix (semantic indexing)*, welche parallel zur dieser Arbeit entwickelt wurde. Sie umfasst alle in dieser Arbeit dargestellten Implementierungen der verschiedenen Algorithmen zur semantischen Indexierung mit expliziten Wissensressourcen. Näheres zur Installation und Benutzung dieser Software können Appendix A und Appendix B entnommen werden.

1. Allgemeine formale Grundlagen

In diesem Kapitel erfolgt eine kurze Zusammenfassung der wichtigsten Formalismen und mathematischen Definitionen, die im weiteren Verlauf dieser Arbeit immer wieder von Bedeutung sein werden. Die Definitionen folgen dabei der Standardnotation aus der Mengenlehre und den formalen Sprachen. Es wird eine grundlegende Vertrautheit mit diesen Formalismen vorausgesetzt.

1.1. Relationen und Hüllen

Definition 1.1.1 (Relation). Seien $A_1, A_2, \ldots, A_n$ Mengen und $n \geq 1$. Eine *n-stellige Relation* ist eine Teilmenge $R \subseteq A_1 \times A_2 \times \cdots \times A_n$ bzw. $R \subseteq \prod_{i=1}^{n} A_i$ auf den Mengen $A_1, A_2, \ldots, A_n$.

Relationen sind somit einfach Mengen aus geordneten n-Tupeln. Hierbei gilt, dass $A_1, A_2, \ldots, A_n$ wiederum (nicht notwendigerweise unterschiedliche) Mengen sind. Gilt insbesondere, dass alle Mengen $A_1, A_2, \ldots, A_n$ gleich sind und setzt man $A := A_1 = A_2 = \cdots = A_n$, dann spricht man auch von einer *n-stelligen Relation auf der Menge A*.

Lemma 1.1.1. Ist $R \subseteq \prod_{i=1}^{n} A_i (n \geq 1)$ eine n-stellige Relation auf den Mengen $A_1, A_2, \ldots, A_n$ und gilt $\langle a_1, a_2, \ldots, a_n \rangle \in R$, dann schreibt man oftmals kürzer: $R(a_1, a_2, \ldots, a_n)$.

Lemma 1.1.2 (Homogene Relation). Eine zweistellige Relation $R \subseteq A \times A$ auf der Menge A wird auch als *(binäre) homogene Relation* auf A bezeichnet.

Definition 1.1.2 (Komposition von Relationen [39]). Seien $R_1 \subseteq A \times A$ und $R_2 \subseteq A \times A$ homogene binäre Relationen auf der Menge A. Die *Komposition* $R_1 \circ R_2$ der beiden Relationen R_1 und R_2 ist definiert als

$$R_1 \circ R_2 := \{\langle a, c \rangle | \exists b : \langle a, b \rangle \in R_1, \langle b, c \rangle \in R_2\}$$

Definition 1.1.3 (N-te Komposition von Relationen [39]). Sei $R \subseteq A \times A$ eine homogene binäre Relation auf der Menge A. Die *n-te Komposition* $R^{(n)}$ der Relation R ist induktiv definiert als

$$R^{(1)} := R$$
$$R^{(n+1)} := R^{(n)} \circ R$$

Die Komposition von zwei Relationen R_1 und R_2 liefert also all jene Tupel $\langle a, c \rangle$, zwischen denen eine transitive Abhängigkeit $\langle a, b \rangle \in R_1, \langle b, c \rangle \in R_2$ besteht. Die n-te Komposition einer Relation bezeichnet die n-fache Komposition einer Relation mit sich selbst.

Definition 1.1.4 (Kette). Sei $R \subseteq A \times A$ eine homogene Relation auf der Menge A und $a_0, a_1, \ldots, a_n \in A$, $n \geq 1$. Die Folge $a_0, a_1, \ldots, a_n \in A$ heißt *R-Kette* der Länge n genau dann, wenn gilt:

$$\forall i = 0, \ldots, n-1 \in \mathbb{N} : R(a_i, a_{i+1})$$

Definition 1.1.5 (Zyklus). Eine R-Kette heißt *R-Zyklus* genau dann, wenn gilt $a_0 = a_n$. Ein R-Zyklus heißt *nicht-trivial*, sofern er mindestens zwei Elemente enthält.

Lemma 1.1.3 (Zyklenfreie Relation). Eine homogene Relation $R \subseteq A \times A$ auf einer Menge A heißt *zyklenfrei*, sofern sie keine nicht-trivialen Zyklen enthält.

Zyklen spielen bei der Berechnung von transitiven Hüllen (vgl. Kapitel 1.1.2) eine Rolle, da die Berechnung von transitiven Hüllen bei zyklenfreien Relationen einfacher zu bewerkstelligen ist.

1.1.1. Eigenschaften von Relationen

Relationen werden oftmals aufgrund ihrer charakteristischen Eigenschaften voneinander unterschieden. Dabei können Relationen stets mehrere Eigenschaften besitzen. Hier werden lediglich die für diese Arbeit wichtigsten Eigenschaften erwähnt.

Definition 1.1.6 (Symmetrische Relation). Eine Relation $R \subseteq A \times A$ auf einer Menge A heißt *symmetrisch* genau dann, wenn gilt

$$\forall a, b \in A : R(a, b) \implies R(b, a)$$

Definition 1.1.7 (Transitive Relation). Eine Relation $R \subseteq A \times A$ auf der Menge A ist *transitiv* genau dann, wenn gilt

$$\forall a, b, c \in A : (R(a, b) \land R(b, c)) \implies R(a, c)$$

Definition 1.1.8 (antisymmetrische Relation). Eine Relation $R \subseteq A \times A$ auf einer Menge A ist *antisymmetrisch* genau dann, wenn gilt

$$\forall a, b \in A : (R(a, b) \land R(b, a)) \implies a = b$$

Bei der Betrachtung von symmetrischen und antisymmetrischen Relationen gilt zu beachten, dass sich diese beiden Eigenschaften von Relationen nicht gegenseitig ausschließen oder bedingen. Eine nicht-symmetrische Relation ist nicht zwangsläufig antisymmetrisch und umgekehrt. Relationen können sowohl nicht-symmetrisch als auch nicht-antisymmetrisch sein.

Lemma 1.1.4 (Transitive, antisymmetrische Relationen). Sei $R \subseteq A \times A$ eine transitive, antisymmetrische Relation auf der Menge A. Daraus folgt direkt, dass R zyklenfrei sein muss.

Beweis. Sei $R \subseteq A \times A$ eine transitive, antisymmetrische Relation auf der Menge A und es existiere ein nicht-trivialer Zyklus

$$R(a_1, a_2), R(a_2, a_3), \ldots, R(a_{n-1}, a_n), R(a_n, a_1), \text{ mit } a_1 \neq a_n$$

Dann folgt aufgrund der Transitivität von R unter anderem auch:

$$R(a_1, a_n), R(a_1, a_n)$$

Dies verletzt die Bedingung der Antisymmetrie (Definition 1.1.8). □

1.1.2. Hüllen

Definition 1.1.9. Eine *Hüllenabbildung* auf einer Menge M ist eine Abbildung $H : \mathcal{P}(M) \mapsto \mathcal{P}(M)$, die jeder Teilmenge $A \subseteq M$ ihre Hülle $H(A) \subseteq M$ so zuordnet, dass stets gilt

$$A \subseteq H(A) \textit{ (Erweiterungseigenschaft)}$$
$$A \subseteq B \implies H(A) \subseteq H(B) \textit{ (Monotonie)}$$
$$H(H(A)) = H(A) \textit{ (Idempotenz)}$$

Im Fall von Hüllenabbildungen binärer symmetrischer oder transitiver Relationen kann man ausnützen, dass die Durchschnittsbildung über eine Menge symme-

trischer und transitiver Relationen selbst wieder eine symmetrische bzw. transitive Relation ergibt. Diese Tatsache soll hier nun am Beispiel symmetrischer Relationen gezeigt werden. Die Argumentation für transitive Relationen folgt dann demselben Schema.

Lemma 1.1.5. Sei $I \subseteq \mathbb{N}$, für jedes $i \in I$ sei $R_i \subseteq M \times M$. Falls alle Relationen R_i symmetrisch sind, so ist auch deren Durchschnitt $\bigcap_{i \in I} R_i$ symmetrisch.

Die Hüllen binärer Relationen R sind nun einfach als Durchschnitt über all jene (symmetrischen, transitiven) Relationen $S \subseteq A \times A$ definiert, die R enthalten. Mit der eben gezeigten Eigenschaft sind die nun folgenden Definitionen für die symmetrischen und transitiven Hüllen von Relationen leicht nachzuvollziehen.

Definition 1.1.10. Sei R eine binäre Relation auf der Menge A. Dann ist die *symmetrische Hülle* von R definiert als:

$$\bigcap \{S \subseteq A \times A | R \subseteq S, S \textit{ symmetrisch}\}$$

Ebenso ist die *transitive Hülle* R^+ von R definiert als:

$$R^+ := \bigcap \{S \subseteq A \times A | R \subseteq S, S \textit{ transitiv}\}$$

Die transitive Hülle einer Relation kann mit Hilfe der Definition 1.1.3 von Kompositionen von Relationen einfacher definiert werden.

Definition 1.1.11 (Transitive Hülle [39]). Die *transitive Hülle* R^+ einer Relation $R \subseteq A \times A$ ist definiert als

$$R^+ := \bigcup_{i \in \{1,2,\ldots,\}} R^{(i)}$$

Abbildung 1.1 verdeutlicht nun abschließend die Hüllenbildung von Relationen. Die mittels der schwarzen Pfeile dargestellte Relation kann durch einfaches Hinzufügen einzelner Beziehungspaare (dargestellt mit den grauen Pfeilen) in deren transitive (Abbildung 1.1a) bzw. symmetrische (Abbildung 1.1b) Hülle überführt werden. Die symmetrische und die transitive Hülle werden bei der Betrachtung der unterschiedlichen Formen von Wissensressourcen eine wichtige Rolle spielen.

1.2. Alphabete, Wörter und Formale Sprachen

Definition 1.2.1 (Alphabet). Ein *Alphabet* ist eine endliche Menge Σ von Zeichen. Die Elemente $\sigma \in \Sigma$ bezeichnet man dabei meist als die *Buchstaben*, *Zeichen* oder *Symbole* des Alphabets.

(a) Transitive Hüllenbildung (b) Symmetrische Hüllenbildung

Abbildung 1.1.: Transitive und symmetrische Hüllenbildung geschieht durch gezieltes Hinzufügen von Verbindungen.

Das Alphabet einer Sprache umfasst somit alle Buchstaben der Sprache. Es werden keine weiteren Annahmen über die einzelnen Zeichen $\sigma \in \Sigma$ gemacht. An dieser Stelle sei aber darauf hingewiesen, dass das verwendete Alphabet aus praktischen Gesichtspunkten auch immer spezielle Zeichen enthalten kann. Dies umfasst unterschiedliche Interpunktionszeichen — wie etwa verschiedene Anführungszeichen «, », Fragezeichen ?, Punkte . oder Ausrufezeichen ! — computertechnische Zeichen — wie etwa einen Zeilenumbruch \n, Leerzeichen ␣ oder bestimmte Metazeichen, die Wortgrenzen und ähnliches markieren. Da die praktische Implementierung des Verfahrens zur semantischen Indexierung auf realen Daten angewendet werden soll, unterstützt es den gesamten Unicode Zeichensatz. Das in der Anwendung verwendete Alphabet und somit auch die verwendeten Alphabete der Automaten der praktischen Anwendung umfassen immer dem gesamten Unicode Zeichensatz und können daher auf unterschiedliche Sprachen und sogar auch parallel auf mehreren Sprachen ohne weitere Anpassungen angewendet werden.

Definition 1.2.2 (Wort). Als *Wort über einem Alphabet* Σ bezeichnet man alle endliche Folgen $w \in \Sigma^*$ über einem Alphabet Σ. Wobei Σ^* die kleenesche Hülle des Alphabets bezeichnet. Das *leere Tupel* bzw. *leere Wort* $\varepsilon := \langle\rangle$ bezeichnet ebenfalls ein Wort über dem Alphabet. Im Weiteren werden Wörter der Länge n als Folge von Zeichen des Alphabets $w = \sigma_1, \ldots, \sigma_n \in \Sigma^*$ dargestellt.

Definition 1.2.3 (Formale Sprache). Jede Teilmenge $\mathcal{L} \subseteq \Sigma^*$ bezeichnet man als *formale Sprache*.

Definition 1.2.4 (Konkatenation). Die *Konkatenation* zweier Wörter $v = \sigma_1, \ldots, \sigma_m$ und $w = \tau_1, \ldots, \tau_n$ ist definiert als $v \circ w := \sigma_1, \ldots, \sigma_m, \tau_1, \ldots, \tau_n$. Anstelle von $v \circ w$ schreibt man oft kürzer vw.

Die Konkatenation zweier Wörter ist assoziativ. Für die Konkatenation mit dem leeren Wort ε gilt stets $\varepsilon \circ w = w \circ \varepsilon = w$.

Aus Definition 1.2.3 geht hervor, dass *jede beliebige* Teilmenge von Σ^* eine (formale) Sprache ist. Sprachen sind in diesem Kontext somit einfach Mengen von Wörtern. Insbesondere definieren auch die noch später folgenden Automaten und Lexika formale Sprachen, die aus all jenen Wörtern bestehen, die sie erkennen bzw. akzeptieren. Bei ihnen handelt es sich um (echte) Teilmengen natürlicher Sprachen, die selbst wiederum Teilmengen der Sternhülle eines Alphabets bilden.

Die Konkatenation mehrerer Wörter eines Alphabets definiert die Kombination dieser Wörter zu einem einzigen.

1.3. Deterministische endliche Automaten

Definition 1.3.1 (Deterministischer endlicher Automat). Ein *deterministischer endlicher Automat (DEA)* Δ wird durch ein fünf-Tupel

$$\Delta = \langle Q, \Sigma, \delta, s, F \rangle$$

repräsentiert, wobei

- Q die endliche Menge der Zustände des Automaten,
- Σ ein Alphabet,
- $\delta : Q \times \Sigma \to Q$ die totale Übergangsfunktion des Automaten,
- $s \in Q$ einen ausgezeichneten Startzustand,
- $F \subseteq Q$ die Menge der Finalzustände des Automaten,

darstellen.

Definition 1.3.2 (Partieller deterministischer endlicher Automat). Ein *partieller deterministischer endlicher Automat (DEA')* $\Delta' = \langle Q, \Sigma, \delta', s, F \rangle$, ist ein Automat im Sinne von Definition 1.3.1 wobei δ' eine partielle Übergangsfunktion $\delta' : Q \times \Sigma \to Q$ ist.

Lemma 1.3.1. Jeder partielle, deterministische endliche Automat $\Delta' = \langle Q, \Sigma, \delta', s, F\rangle$ lässt sich durch Einfügung eines speziellen, nicht-finalen *Fallenzustands* $\theta\neg \in Q$ in einen äquivalenten (siehe Definition 1.3.6) deterministischen Automaten $\Delta = \langle Q \cup \{\theta\}, \Sigma, \delta, s, F\rangle$ überführen. Für die totale Übergangsfunktion δ des Automaten gilt dann für $\sigma \in \Sigma$ und $q \in Q$:

$$\delta(\langle\sigma, q\rangle) := \begin{cases} \theta & \text{falls } q = \theta \vee \langle\sigma, q\rangle \notin Def(\delta') \\ \delta'(\langle\sigma, q\rangle) & \text{ansonsten} \end{cases}$$

Die Definition des Fallenzustands im Automaten erleichtert die folgenden algorithmischen Betrachtungen, da nicht weiter zwischen partiellen und (totalen) Automaten unterschieden werden muss, da durch die Einführung eines Fallenzustands alle partiellen Automaten in totale Automaten umgewandelt werden können. Alle *illegalen* Übergänge im Automaten führen immer in diesen nicht-finalen Fallenzustand. Von dem Fallenzustand aus führen allen Übergänge immer wieder zu dem Fallenzustand selbst zurück. Die meisten Darstellungen von Automaten verzichten auf die Darstellung von Übergängen hin zu dem Fallenzustand des Automaten. Falls in einer Abbildung von einem bestimmen Zustand $q \in Q$ aus kein Übergang mit $\sigma \in \Sigma$ dargestellt ist, existiert implizit immer ein entsprechender Übergang mit $\delta(\sigma) = \theta$ in den Fallenzustand.

Die meisten der in dieser Arbeit betrachteten Automaten sind partiell. Da jeder partielle Automat durch Einfügen eines Fallenzustands einfach in einen Automaten mit totaler Übergangsfunktion überführt werden kann, wird diese Unterscheidung von Automaten im weiteren Verlauf dieser Arbeit nicht weiter beachtet.

Im Hinblick auf die Implementierung deterministischer Automaten, die bei der Implementierung der semantischen Indexierung mit expliziten Wissensressourcen eine Rolle spielen, sei darauf hingewiesen, dass — gerade auch bei der Verwendung großer Alphabete — totale Automaten oftmals viel größer sind als partielle, vor allem wenn man die Übergänge des Fallenzustands explizit mit speichert. Aus diesem Grund sind die in dieser Arbeit betrachteten Automaten als partielle Automaten implementiert und fehlende Übergänge werden von der Implementierung speziell behandelt.

Definition 1.3.3 (Verallgemeinerte Übergangsfunktion). Bezeichne $\Delta = \langle Q, \Sigma, \delta, s, F, \theta\rangle$ einen deterministischen endlichen Automaten, dann ist die *verallgemeinerte Übergangsfunktion* $\delta^* : Q \times \Sigma^* \to Q$ mit $q \in Q, w \in \Sigma^*$ rekursiv definiert als:

$$\delta^*(q, \varepsilon) := q$$
$$\delta^*(q, w\sigma) := \delta(\delta^*(q, w), \sigma)$$

Definition 1.3.4 (Akzeptanz von Wörtern). Seien $w \in \Sigma^*$ ein Wort und $\Delta = \langle Q, \Sigma, \delta, s, F, \theta \rangle$ ein deterministischer endlicher Automat. Der Automat Δ *akzeptiert ein Wort* w genau dann, wenn gilt:

$$\delta^*(s, w) \in F$$

Definition 1.3.5 (Sprache eines Automaten). Mit der *(formalen) Sprache* $\mathcal{L}(\Delta) \in \Sigma^*$ *eines Automaten* $\Delta = \langle Q, \Sigma, \delta, s, F, \theta \rangle$ wird die von Δ akzeptierte Sprache $\mathcal{L}(\Delta) := \{w \in \Sigma^* | \delta^*(s, w) \in F\}$ bezeichnet, die alle Wörter beinhaltet, die vom Automaten Δ akzeptiert werden.

Die verallgemeinerte Übergangsfunktion $\delta^*(q, w)$ stellt einfach die Anwendung der Übergangsfunktion δ auf die Buchstaben eines Wortes in Lesereihenfolge dar. Ein Automat akzeptiert ein Wort w immer genau dann, wenn ein Wort Teil der Sprache des Automaten ist: $w \in \mathcal{L}(\Delta)$.

Definition 1.3.6 (Äquivalenz von Automaten). Zwei Automaten Δ_1 und Δ_2 sind äquivalent genau dann, wenn sie über eine äquivalente Sprache verfügen:

$$\Delta_1 = \Delta_2 : \Longleftrightarrow \mathcal{L}(\Delta_1) = \mathcal{L}(\Delta_2)$$

Definition 1.3.7 (Rechtsprache [11]). Sei $\Delta = \langle Q, \Sigma, \delta, s, F, \theta \rangle$ ein deterministischer endlicher Automat. Die Rechtssprache $\vec{\mathcal{L}}(q)$ eines Zustands $q \in Q$ ist rekursiv definiert als:

$$\vec{\mathcal{L}}(q) = \bigcup \{a \circ \vec{\mathcal{L}}(\delta(q, a)) | a \in \Sigma\} \cup \begin{cases} \{\varepsilon\} & \text{falls } q \in F \\ \emptyset & \text{ansonsten} \end{cases}$$

Definition 1.3.8 (Äquivalenz von Zuständen). Sei $\Delta' = \langle Q, \Sigma, \delta', s, F, \theta \rangle$ ein deterministischer endlicher Automat. Zwei Zustände $p, q \in Q$ des Automaten sind äquivalent genau dann, wenn sie über die gleiche Rechtssprache verfügen:

$$p \equiv q : \Longleftrightarrow \vec{\mathcal{L}}(p) = \vec{\mathcal{L}}(q)$$

Lemma 1.3.2. Sei $\Delta = \langle Q, \Sigma, \delta, s, F, \theta \rangle$ ein deterministischer endlicher Automat. Zwei Zustände $p, q \in Q$ des Automaten sind äquivalent genau dann, wenn gilt:

- $p \in F \iff q \in F$
- $\forall \sigma \in \Sigma : \delta(p, \sigma) \equiv \delta(q, \sigma)$

1.4. Die Levenshteindistanz

Eine weit verbreitete Maßzahl zur Berechnung von Wortabständen ist der sogenannte *Levenshteinabstand* d_L oder Editierabstand [35]. Er wird allgemein definiert als die minimale Anzahl elementarer Editieroperationen, die notwendig sind, um ein Wort w in ein anderes Wort v zu überführen. Im einfachen Fall umfassen diese elementaren Editieroperationen Einfügungen, Löschungen und Transpositionen auf Zeichenebene.

Beispiel 1.4.1. Der einfache Levenshteinabstand zwischen den beiden Wörtern *Tier* und *Tor* ist $d_L(Tor, Tier) = 2$. Die beiden zugehörigen Editieroperationen sind zum Beispiel die Transposition von *i* in *Tier* nach *o* in *Tor* und die Löschung von *e* in *Tier*.

Die Berechnung des Levenshteinabstands kann auf einfache Weise nach [46] erfolgen. Es gibt eine Vielzahl von Erweiterungen dieses einfachen Basisalgorithmus, die weitere Editieroperationen wie Verschmelzungen und Aufspaltungen von Buchstaben betrachten und auch unterschiedliche Gewichtungen für besonders (un-) wahrscheinliche Editieroperationen erlauben.

1.4.1. Nicht deterministischer Automat zur Erkennung benachbarter Wörter

In [14] wird ein nicht deterministischer Automat beschrieben, der für ein bestimmtes Wort w all jene Wörter u akzeptiert, die eine Levenshteindistanz von nicht mehr als k zu w besitzen: $d_L(u, w) \leq k$. Dieser Automat soll hier kurz näher erläutert werden, da er einerseits als Grundlage für ein schnelles Verfahren zur approximativen Suche auf Wörterbüchern dient [54], andererseits auch das Verfahren aufzeigt, welches zur fehlertoleranten Suche auf Mehrwortverbindungen in einem Dokument verwendet werden kann (vgl. Kapitel 6).

Abbildung 1.2 zeigt exemplarisch einen solchen Automaten, der alle Wörter mit einem Editierabstand $k \leq 2$ zum Wort *essen* erkennt. Der Automaten ist aus $k + 1$ Schichten aufgebaut, wobei die horizontalen Übergänge einer Schicht von rechts nach links jeweils genau das Wort erkennen, aus dem der Automat aufgebaut wurde. Jede Schicht entspricht immer einem festen Editierabstand k[1].

Alle Zustände — mit Ausnahme der Finalzustände — haben jeweils drei unterschiedliche Typen von Übergängen, wobei jeder Typ als eine atomare Editieroperation des Levenshteinabstands angesehen werden kann. Senkrechte Übergänge

[1] Hierbei ist zu beachten, dass in diesem Fall die unterste Schicht einen Editierabstand von 0 symbolisiert — akzeptierte Wörter in dieser Schicht entsprechen genau dem Ausgangswort mit einer Levenshteindistanz von 0.

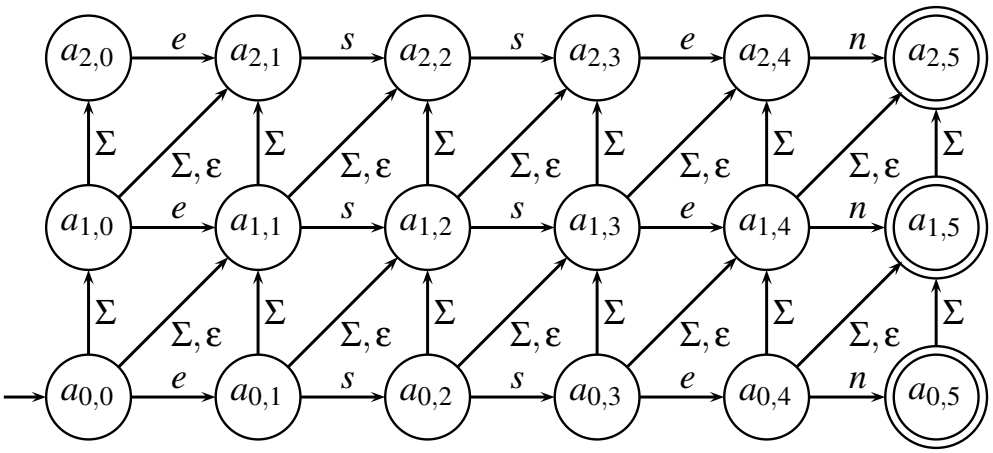


Abbildung 1.2.: Nichtdeterministischer Levenshteinautomat, der alle Wörter akzeptiert, die eine Levenshteindistanz $k \leq 2$ zu dem Wort *essen* aufweisen. Die mit Σ gekennzeichneten Übergänge symbolisieren Übergänge, die mit jeden Buchstaben des Alphabets möglich sind (frei nach [47]).

von einem Zustand $a_{i,j}$ zu einem Zustand $a_{i+1,j}$ entsprechen dabei dem Einfügen eines Buchstabens. Es wird ein Zeichen in u gelesen, während kein Zeichen in w gelesen wird. Horizontale Übergänge entsprechen dem Akzeptieren eines Zeichens, welches sowohl in u als auch in w enthalten ist. Die diagonalen Übergänge entsprechen dann entweder der Löschung eines Zeichens bei Übergängen mit ε oder — bei beliebigen Σ-Übergängen der Substitution eines Zeichens in u mit einem Zeichen in w. Sofern der Automat ein Wort u akzeptiert, gilt einerseits natürlich $d_L(w, u) \leq k$, andererseits kann $d_L(w, u)$ aus den aktiven Finalzuständen abgelesen werden. Der Wortabstand entspricht der Schicht i des untersten aktiven Finalzustands $a_{i,j}$ (vgl. [47]).

Der Automat weist eine sehr homogene Struktur auf. In [68] wird eine Methode vorgestellt, diesen Automaten durch bit-parallele Methoden effizient zu simulieren. Es ist sogar möglich, aus der Struktur dieser nicht deterministischen Automaten einen deterministischen universellen Levenshteinautomaten herzuleiten [54].

1.5. Explizite Wissensressourcen

Explizite Wissensressourcen in dieser Arbeit repräsentieren das Wissen, das bei der semantischen Indexierung zur Verfügung steht. Sie stellen die zentrale Struktur bei der semantischen Indexierung mit expliziten Wissensressourcen dar.

In der Literatur [26, 27, 22] werden unterschiedliche Formen von Ontologien und semantischen Netzen unterschieden. Die nachfolgende Definition versucht die Darstellung zu vereinheitlichen und legt einen besonderen Fokus auf die Verbindung von natürlichsprachlichen Ausdrücken mit abstrakten Konzepten.

Definition 1.5.1 (Explizite Wissensressource). Eine *explizite Wissensressource* ist ein Tupel $\mathcal{W} = \langle \mathcal{W}_K, \mathcal{W}_R, \mathcal{W}_{Lex}, \kappa \rangle$, wobei

- $\mathcal{W}_K$ eine nicht leere Menge von Konzepten,
- $\mathcal{W}_R$ eine Menge von zweistelligen Relationen auf $\mathcal{W}_K$,
- $\mathcal{W}_{Lex} \subseteq \Sigma^*$ eine endliche formale Sprache auf dem Alphabet Σ und
- $\kappa : \mathcal{W}_{Lex} \to \mathcal{W}_K$ eine totale Abbildung von $\mathcal{W}_{Lex}$ nach $\mathcal{W}_K$

sind.

$\mathcal{W}_K$ heißt *Konzeptmenge*, $\mathcal{W}_R$ *Relationsmenge*, $\mathcal{W}_{Lex}$ Lexikon und κ *Konzeptzuweisungen* der Wissensressource $\mathcal{W}$. Die Wörter $w \in \mathcal{W}_{Lex}$ heißen auch *Lexikoneinträge* der Wissensressource $\mathcal{W}$.

Definition 1.5.2 (Bildmenge von Konzepten). Sei $\mathcal{W} = \langle \mathcal{W}_K, \mathcal{W}_R, \mathcal{W}_{Lex}, \kappa \rangle$ eine Wissensressource. Die *Bildmenge* $\mathcal{W}_R(K)$ *eines Konzepts* $K \in \mathcal{W}_K$ bezeichnet alle Konzepte $K' \in \mathcal{W}_K$ der Wissensressource, mit denen K über irgendeine Relation $R \in \mathcal{W}_R$ verbunden ist:

$$\mathcal{W}_R(K) := \{K' | K' \in \mathcal{W}_K \wedge \exists R \in \mathcal{W}_R : R(K, K')\}$$

Die Bildmenge von Konzepten bezeichnet alle diejenigen Konzepte, die von einem bestimmten Konzept referenziert werden. Die genaue Relation, unter der die Konzepte miteinander verbunden sind, spielt dabei keine Rolle. Gilt insbesondere $\mathcal{W}_R = \{R\}$, wie es unter anderem auch für *EFGT-Netze* (vgl. Kapitel 4.3) gilt, kann man auch einfacher $R(K)$ anstelle von $\mathcal{W}_R(K)$ schreiben.

Die expliziten Wissensressourcen aus Definition 1.5.1 stellen das explizite Weltwissen dar, das bei der semantischen Indexierung mit expliziten Wissensressourcen verwendet wird. Die Konzepte der Wissensressource $\mathcal{W}_K$ sind eindeutige Themen und Konzepte, die unter verschiedenen Relationen der Relationsmenge $\mathcal{W}_R$ miteinander verbunden sind. Eben diese Verbindungen zwischen den Konzepten

werden bei der semantischen Indexierung verwendet, um den Index um das in den Relationen abgebildete Wissen zu erweitern. Die unterschiedlichen Relationen in $\mathcal{W}_R$ bilden dabei verschiedene Formen von Zugehörigkeiten der Konzepte ab. Sie sind die Prädikate, unter denen die Konzepte der expliziten Wissensressourcen verbunden sind und modellieren die logisch-semantischen Zusammenhänge der Themen und Konzepte der Wissensressourcen untereinander. Die Relationen bilden somit das eigentliche semantische Wissen der Ressourcen.

Natürlichsprachliche Texte kodieren ihre enthaltenen Konzepte über eine Reihe von mehr oder weniger komplexen, natürlichsprachlichen Ausdrücken. Um die abstrakten Konzepte der Wissensressourcen in natürlichsprachlichen Texten identifizieren zu können, enthalten die expliziten Wissensressourcen ein Lexikon $\mathcal{W}_{Lex}$, welches die natürlichsprachlichen Ausdrücke enthält, die zum Auffinden der Konzepte in den Eingabedokumenten verwendet werden. Jeder Lexikoneintrag in $\mathcal{W}_{Lex}$ ist über κ eindeutig einem Konzept der Wissensressource zugeordnet. Wann immer ein Lexikoneintrag in einem Text identifiziert wird, kann er so einfach seinem abstrakten Konzept zugeordnet werden, welches dann bei der Indexierung weiter verwendet wird.

Beispiel 1.5.1. Eine explizite Wissensressource $\mathcal{W} = \langle \mathcal{W}_K, \mathcal{W}_R, \mathcal{W}_{Lex}, \kappa \rangle$ bestehe aus der Konzeptmenge $\mathcal{W}_K = \{O, S, V, A, D\}$, der Relationsmenge $\mathcal{W}_R = \{R\}$ aus der Relation $R = \{\langle V, S\rangle, \langle A, O\rangle, \langle O, D\rangle, \langle S, D\rangle\}$, dem Lexikon $\mathcal{W}_{Lex} = \{$*Ankylosaurus*, *Velociraptor*, *Ornithischia*, *Saurischia*, *Dinosauria*$\}$ und den Konzeptzuweisungen $\kappa = \{\langle$*Dinosauria*$, D\rangle, \langle$*Saurischia*$, S\rangle, \langle$*Ornithischia*$, O\rangle, \langle$*Velociraptor*$, V\rangle, \langle$*Ankylosaurus*$, A\rangle\}$.

Diese Wissensressource bildet die vereinfachte taxonomische Einteilung des Echsenbeckensauriers (Saurischia) *Velociraptor* und des Vogelbeckensauriers (Ornithischia) *Ankylosaurus* ab. R entspricht dabei in etwa dem Prädikat *„ist ein"* bzw. *„gehört zur Familie der"*. So bildet zum Beispiel das Tupel $\langle V, O\rangle \in R$ die semantische Bedeutung *„Velociraptoren gehören zur Familie der Ornithischia"* ab.

Die Konzeptmenge $\mathcal{W}_K$ der Wissensressource enthält eindeutige, abstrakte Konzepte. Das Lexikon $\mathcal{W}_{Lex}$ enthält die verschiedenen natürlichsprachlichen Ausrücke der Konzepte. Die Konzeptzuweisungen κ verbinden die natürlichsprachlichen Ausdrücke im Lexikon eindeutig mit den Konzepten der Wissensressource, wobei mehrere unterschiedliche Lexikoneinträge demselben Konzept, einzelne Lexikoneinträge aber niemals mehreren Konzepten zugeordnet seien können. Wird zum Beispiel in einem Dokument der natürlichsprachliche Ausdruck *„Ankylosaurus"* identifiziert, kann er sogleich über κ dem Konzept $A \in \mathcal{W}_K$ zugeordnet werden.

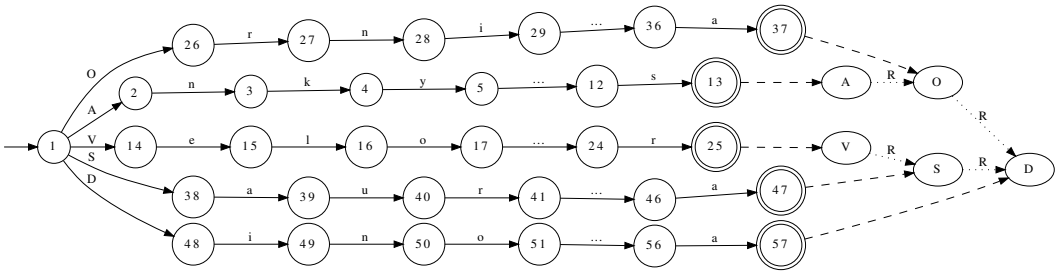


Abbildung 1.3.: Der $\mathcal{W}$-Automat aus Beispiel 1.5.1. Die Finalzustände des Automaten sind über die Typisierungsfunktion t (gestrichelte Linien) mit den Konzepten der expliziten Wissensressource verbunden, welche ihrerseits untereinander über die Relation $R \in \mathcal{W}_R$ der Wissensressource $\mathcal{W}$ (gepunktete Linien) verbunden sind.

Definition 1.5.3 ($\mathcal{W}$-Automat). Sei $\mathcal{W} = \langle \mathcal{W}_K, \mathcal{W}_R, \mathcal{W}_{Lex}, \kappa \rangle$ eine Wissensressource. Ein *$\mathcal{W}$-Automat* ist ein Tupel der Form $\langle \Sigma, Q, s, \delta, F, \theta, t \rangle$, wobei gilt, dass

- $\mathcal{A} = \langle \Sigma, Q, s, \delta, F, \theta \rangle$ ein deterministischer endlicher Automat und
- $t : F \to \mathcal{W}_K$ eine Typisierungsfunktion von den Finalzuständen des Automaten auf die Konzepte der Konzeptmenge

sind. Für jeden String $w \in \Sigma^*$ gilt:

$$w \in \mathcal{W}_{Lex} \iff w \in \mathcal{L}(\mathcal{A})$$
$$w \in \mathcal{W}_{Lex} \implies \kappa(w) = t(\delta^*(s, w))$$

Der $\mathcal{W}$-Automat aus Definition 1.5.3 dient bei der semantischen Indexierung mit expliziten Wissensressourcen vor allem der Speicherung des Lexikons $\mathcal{W}_{Lex}$ und der Suche nach Lexikoneinträgen in den Eingabedokumenten der semantischen Indexierung. Er akzeptiert genau die Wörter, die Elemente des Lexikons der expliziten Wissensressourcen sind. Die Finalzustände des Automaten sind über seine Typisierungsfunktion t mit den Konzepten der expliziten Wissensressource eindeutig verbunden, so dass jeder Finalzustand des Automaten eindeutig auf genau das Konzept abgebildet werden kann, welches die mit diesem Finalzustand akzeptierten Lexikoneinträge repräsentiert.

Bei der semantischen Indexierung mit expliziten Wissensressourcen wird der $\mathcal{W}$-Automat aus den Lexikoneinträgen des Lexikons der expliziten Wissensressourcen aufgebaut und mit deren Konzepten verbunden. Er wird zur effizienten Suche nach Lexikoneinträgen auf den Eingabedokumenten und somit zur Identifizierung der zugrundeliegenden Konzepte in den Dokumenten verwendet (vgl.

Kapitel 5).

2. Linguistische Aspekte

Die semantische Indexierung verarbeitet natürlichsprachlichen Text. Wie auch bei anderen Indexierungsverfahren, müssen bei der Indexierung mit expliziten Wissensressourcen verschiedene sprachliche Eigenarten gehandhabt werden, auf die in diesem Kapitel eingegangen wird. Während bei klassischen Verfahren zur Verringerung der Indextermmenge verschiedene Stemming- und Lemmatisierungsverfahren angewendet werden, arbeitet die hier dargestellte semantische Indexierung mit ausführlichen Vollformenlexika, um auch verschiedene Derivations- und Flektionsformen der Lexikoneinträge behandeln zu können. Diese Lexika können darüber hinaus nicht nur Einzelwörter, sondern auch explizit komplexe Mehrwortverbindungen beinhalten. Dies führt insbesondere auch zu einer besseren Handhabbarkeit von Fehlern und zu einer Vermeidung verschiedener Arten von Ambiguitäten.

2.1. Behandlung natürlichsprachlicher Texte

Anwendungen, die unterschiedliche Repräsentationen von Wissen verwenden, müssen — anders als Anwendungen, die sich nur mit der Repräsentation von Wissen beschäftigen — früher oder später Texte verarbeiten, um aus diesen Informationen zu erhalten, die dann — je nach Anwendung — weiter verarbeitet werden.

Somit muss gerade auch die semantische Indexierung mit expliziten Wissensressourcen mit natürlichsprachlichen Texten umgehen können. Die verarbeiteten Texte sind dabei meist von Menschen erzeugt. Es sollte aber nicht vergessen werden, dass in der heutigen Zeit — gerade auch in Hinblick auf das Internet und andere mobile Serviceleistungen — weitere Erzeuger von natürlichsprachlichen Texten in Frage kommen. Diese maschinell erzeugten Texte umfassen unter anderem automatisch erstellte Textzusammenfassungen, maschinell übersetzte Texte aus anderen Sprachen oder auch Texte, die mit einer maschinellen Sprach- oder Zeichenerkennung aus analogen Ursprungsdaten erzeugt wurden. In solchen automatisch erzeugten Texten treten zusätzlich zu sprachlichen Phänomenen auch noch weitere Probleme auf, wie zum Beispiel unterschiedliche (historisch bedingte) Rechtschreibungen, fremdsprachliche Ausdrücke oder Rechtschreibfehler.

Will man Wissensressourcen zur semantischen Indexierung verwenden, muss einerseits die Frage geklärt werden, inwieweit die vorhandenen Wissensressourcen direkt dazu geeignet sind, sie zur semantischen Indexierung einzusetzen und wie sie gegebenenfalls angepasst werden müssen, um eine semantische Indexierung von natürlichsprachlichen Texten zu ermöglichen. Andererseits müssen eine Vielzahl linguistischer Phänomene betrachtet werden, die bei dem Aufbau der Lexika sowie bei der nachfolgenden semantischen Indexierung eine Rolle spielen.

Die häufigsten Probleme, die bei der Behandlung von natürlichsprachlichen Texten auftreten, sind neben Flexion und Derivation[1] Ambiguitäten, komplexe Mehrwortverbindungen und Schreibvarianten oder Fehler jeglicher Art. Diese Aspekte und deren Behandlung bei der Verarbeitung von Texten werden im weiteren Verlauf nun dargestellt. Es werden jeweils auch Ansätze zur Lösung dieser Problemen diskutiert, die dem weiteren Verlauf dieser Arbeit als Basis dienen.

2.2. Textanalyse mit Wissensressourcen

Wie bereits erwähnt, ist die Grundidee bei der semantischen Indexierung von Texten, dass in den zu indexierenden natürlichsprachlichen Texten eine Reihe von abstrakten Ideen oder Konzepten identifiziert werden können. Texte zu unterschiedlichen Themenbereichen weisen stark voneinander abweichende Themen und Konzepte auf, wohingegen Texte zu ähnlichen Themen ähnliche Konzepte und Ideen enthalten.

Die Konzepte selbst sind meist nicht direkt im Text enthalten sondern über verschiedene sprachliche Ausdrücke realisiert. Um Texte mit Wissensressourcen verarbeiten zu können, müssen zuallererst die in den Wissensressourcen verwendeten Konzepte im Text identifiziert werden. Es braucht also eine Zuordnung von sprachlichen Ausdrücken zu den Konzepten der Wissensressourcen.

Die Abbildung natürlichsprachlicher Ausdrücke auf die Konzepte der Wissensressource kann über ein einfaches *Lexikon* geschehen. Jeder Eintrag des Lexikons verweist auf ein Konzept der Wissensressource (siehe dazu insbesondere die Definitionen 1.5.1 und 1.5.3). Die Einträge des Lexikons können dabei nicht nur einzelne Wörter umfassen, sondern auch komplexere sprachliche Ausdrücke und Phrasen aus mehreren Einzelwörtern.

Im günstigsten Fall sind die Zuordnungen zwischen den Lexikoneinträgen des Lexikons und den Konzepten der Wissensressource bereits direkt in den Wissensressourcen selbst kodiert. Für Wissensressourcen ohne solche direkten Zuordnungen können die benötigten Zuordnungen auch im Nachhinein noch von außen an die Ressource angefügt werden.

[1] Diese Aspekte sind natürlich stark von den behandelten natürlichen Sprachen abhängig.

Die Erstellung der Lexikoneinträge kann — ebenso wie die Erstellung der eigentlichen Wissensressourcen — nur bedingt automatisiert werden. Oftmals können die natürlichsprachlichen Vorzugsnamen — die ja selbst meist so gewählt werden, dass sie die Konzepte möglichst eindeutig beschreiben — als Lexikoneinträge ausgewählt werden. Ausgehend von diesen können dann weitere, möglichst eindeutige Ausdrücke mit in das Lexikon aufgenommen werden. Dabei erschweren verschiedene linguistische Phänomene die Erstellung dieser Lexikoneinträge. Sie müssen bei der Erstellung der lexikalischen Hilfsressourcen beachtet und möglichst einheitlich aufgelöst werden. Diese Phänomene werden nun in den folgenden Abschnitten genauer diskutiert.

2.3. Flexion und Derivation

Flexion und Derivation bezeichnen linguistische Phänomene, die dazu führen, dass semantisch gleiche Wörter je nach syntaktischer Notwendigkeit unterschiedliche Oberflächenformen ausprägen. *Flexion* bezeichnet die Veränderung der Oberflächenform eines Wortes (etwa durch Suffigierung oder Affigierung), um verschiedene grammatische Funktionen im Satz zu markieren. Dabei bleibt die Wortart erhalten. *Derivation* dagegen dient der Erzeugung neuer Wortformen durch die Verbindung eines bestehenden Wortes des Wortschatzes mit einem abstrakten Wortteil ohne konkrete lexikalische Bedeutung. Hierbei bleibt die Wortart im Gegensatz zur Flexion nicht erhalten. Die *Komposition* ist ein weiteres vergleichbares Phänomen, welches ein wenig an die Derivation erinnert. Im Gegensatz zur Derivation werden bei der Komposition zwei oder mehr Wörter mit lexikalischen Bedeutungen verbunden, um ein neues Wort zu schaffen [36, S. 68ff].

Beispiel 2.3.1. Ein Beispiel für Flexion ist die Pluralbildung von Substantiven. Aus *Haus* wird im Plural *Häuser*, aus *city* wird *cities*. Die Wortart bleibt unverändert, während sich die Oberflächenform des Worts ändert.

Bei der Derivation dagegen, ändert sich nicht nur Oberflächenform des Worts, sondern auch die Wortart. Aus dem Adjektiv *frei* und dem Suffix *heit* wird das Substantiv *Freiheit* gebildet.

All diese Phänomene sind *sprachabhängig*. Sie tauchen in unterschiedlichen natürlichen Sprachen mit unterschiedlicher Häufigkeit und in unterschiedlichen Ausprägungen auf. So existiert zum Beispiel im Englischen eine Kompositabildung, die mit der deutschen vergleichbar ist. Bei bestimmten Wortverbindungen dagegen, werden die einzelnen Wortbestandteile der Komposita aber nach verschiedenen Regeln entweder mit einem Leerzeichen oder einem Bindestrich (oder einer Mi-

schung der beiden) voneinander getrennt und liefern ein weiteres Argument für die Aufnahme Mehrwortverbindungen in die Lexika [38, S. 67ff].

Diese natürlichsprachlichen Phänomene erschweren die Erkennung von Konzepten einer Wissensressource in Texten. Die unterschiedlichen Realisierungen einzelner lexikalischer Zuordnungen müssen bei der Indexierung zusätzlich mit berücksichtigt werden. Im klassischen IR bei der Indexierung von Texten wurden diese Phänomene bereits auf verschiedene Weise und mit verschiedenen Vor- und Nachteilen behandelt [66, S. 146].

2.3.1. Stemming

Eine populäre, wenn auch grobe Art, die unterschiedlichen Ausprägungen von Wörtern in den Griff zu bekommen und dafür zu sorgen, dass semantisch ähnliche Wörter gleichermaßen im Index gefunden werden können, ist das sogenannte *Stemming*. Dieses Verfahren bearbeitet Wortformen nach einer Reihe unterschiedlicher sprachabhängiger Regeln. Hierbei werden im Allgemeinen verschiedene Derivations- und Flexionssuffixe von den Wörtern entfernt [37] und so in ihre sogenannte *Wurzelform*[2] überführt [66, S. 146].

Dieses Verfahren ist vergleichsweise schnell und gut dazu in der Lage, verschiedene Wortformen mit einer gleichen oder ähnlichen Wurzel zu einem eindeutigen Wort zu verschmelzen. Es bestehen verschiedene Algorithmen und Implementierungen, die es ermöglichen, das Stemming einfach in ein System zur Indexierung von Text einzubauen [45]. Durch Anwendung desselben Stemmingalgorithmus auf die späteren Suchanfragen können die reduzierten Wörter wieder im Index aufgefunden werden. Ein Vorteil des Stemmings, der gerade in klassischen Anwendungen des IR eine wichtige Rolle spielt, ist die starke Verkleinerung der Termmenge und die damit direkt verbundene Verkleinerung der Indexgröße.

Wie bereits angedeutet, ist das Stemming eine sehr grobe Art, den Text zu behandeln. Es arbeitet allein auf der Oberflächenform der Token im Text und beachtet weder den Kontext noch die grammatische Bedeutung der einzelnen Token. Je nach Sprache funktionieren unterschiedliche Stemmingalgorithmen unterschiedlich gut. Auch treten bei Eigennamen bei dieser Form der Textverarbeitung eine Reihe von Problemen auf [66, S. 147]. Ebenso kann das Stemming zu *künstlichen Ambiguitäten*, also Ambiguitäten, die nicht im ursprünglichen Text vorhanden sind, sondern erst durch das Stemming eingeführt werden, führen.

Wie beim klassischen IR kann Stemming auch bei der semantischen Indexierung angewendet werden. Dazu müssen die Dokumente, die Lexikoneinträge und

[2] Diese Wurzelform hat im Allgemeinen nichts nichts mit der linguistische Wurzel eines Wortes zu tun. Es kann sich dabei streng genommen auch um eine nur von dem System lesbare Folge von von willkürlich gewählten Zeichen handeln.

die einzelnen Token von Suchanfragen gleichermaßen mit einem geeigneten Stemmingalgorithmus bearbeitet werden. Dies kann vollkommen automatisch geschehen und erfordert sowohl beim Aufbau der Wissensressource und des Lexikons als auch bei der Indexierung und der Bearbeitung der Suchanfragen keinen manuellen Aufwand. Das Stemming führt dabei ebenfalls zu einer starken Verkleinerung der Menge der Lexikoneinträge und erlaubt es eine Vielzahl sprachlich ähnlicher Ausdrücke ohne weitere Beachtung etwaiger Flexionsformen uniform zu bearbeiten.

Neben der Einführung weiterer Ambiguitäten, die in Abschnitt 2.4 noch genauer erläutert werden und der starken Sprachabhängigkeit, wirkt sich für die semantische Indexierung allerdings vor allem auch die Schwäche des Stemmings bei Eigennamen nachteilig aus. Bei vielen spezialisierten Wissensressourcen, wie zum Beispiel der Ontologie zum ersten Weltkrieg (vgl. Abschnitt 4.5.2), bilden Eigennamen von Personen und Orten einen Großteil der im Lexikon der Wissensressourcen abgespeicherten Bezeichner. Gerade diese Eigennamen sollen bei der Indexierung möglichst sicher und eindeutig in den Texten identifiziert werden können.

Während es grundsätzlich möglich ist, Stemming auf das hier vorgestellte Verfahren zur semantischen Indexierung anzuwenden — nämlich indem das Stemming über spezielle *Textnormalisierer* (vgl. Kapitel 2.7 und 5.4.1) sowohl auf die Texte und Suchanfragen als auch auf die Lexikoneinträge der Wissensressource angewendet werden — verzichtet das hier dargestellte Verfahren auf Stemming, um insbesondere auch die Anzahl möglicher Ambiguitäten zu verringern und um Eigennamen möglichst sicher behandeln zu können.

2.3.2. Lemmatisierung

Eine weitere Technik zur Behandlung der hier dargestellten Phänomene ist die sogenannte *Lemmatisierung*. Diese Technik versucht die Token auf ihr Lemma abzubilden. Im Gegensatz zum Stemming wird bei der Lemmatisierung zumeist ein Vollformenlexikon der entsprechenden Sprache benötigt. Außerdem muss der Kontext der Token im Satzgefüge mitberücksichtigt werden, da die Zuordnung einzelner Flektionsformen auf ihre Lemmata im Allgemeinen nicht eindeutig ist. So ist selbst bei häufigen Wörtern wie z.B. *einen* ohne Kontext nicht zu entscheiden, ob es sich um ein Verb oder den unbestimmten Artikel handelt.

Es gibt eine Vielzahl unterschiedlicher Ansätze und Werkzeuge, die eine automatische Lemmatisierung ermöglichen. Ein weit verbreitetes System zur Lemmatisierung und zum POS-Tagging ist der sogenannte *TreeTagger* [49].

Neben der Kontextabhängigkeit stellt ebenso wie beim Stemming auch die Sprachabhängigkeit ein Nachteil dieses Verfahrens dar. Gerade auch die Ontologie „Erster Weltkrieg“ (vgl. Kapitel 4.5.2) enthält Lexikoneinträge zu mehreren Sprachen und soll auch auf Texte in unterschiedlichen Sprachen angewendet werden können.

Somit kann die Lemmatisierung im Gegensatz zum Stemming nicht so einfach automatisch auf die semantische Indexierung angewendet werden. Gerade Suchanfragen und die Lexikoneinträge weisen im Allgemeinen zu wenig Kontextinformationen auf, um eine sichere automatische Lemmatisierung und Spracherkennung auf diesen durchzuführen. Somit müssten die bestehenden Ressourcen teilweise manuell angepasst werden.

Aus diesen Gründen verwendet das hier betrachtete Verfahren keine Lemmatisierung. Vielmehr wird ein umgekehrter Ansatz gewählt, um die Erkennung von Konzepten der Wissensressourcen in den Dokumenten zu ermöglichen. Um die Menge der Token zu reduzieren, werden dabei nicht die Eingabetexte vorverarbeitet, sondern es werden Vollformenlexika verwendet, die möglichst viele sprachliche Variationen der zu erkennenden Textbausteine abdecken.

2.3.3. Vollformenlexika

Eine Möglichkeit unterschiedliche Flexionsformen in natürlichsprachlichen Texten zu handhaben ist, nicht einzelne Wortformen auf ihr Lemma abzubilden, sondern umgekehrt die Formen der entsprechenden Paradigmen explizit im Lexikon zu speichern. Hierbei werden nicht nur die Paradigmen einzelner Wörter, sondern auch die verschiedenen Flexionsformen ganzer Phrasen explizit und möglichst vollständig im Lexikon abgespeichert.

Dieses Verfahren vergrößert die Menge der Lexikoneinträge deutlich und erschwert zusätzlich die manuelle Erzeugung der Lexika, da für einzelne Wörter oder Phrasen oftmals viele Erzeugungen eingefügt werden müssen.

Bei diesem Vorgehen wird einerseits eine Implementierung des Lexikons benötigt, die in der Lage ist, gleichzeitig viele Lexikoneinträge beliebiger Länge zu speichern. Andererseits muss die Implementierung auch ein schnelles Verfahren bereitstellen, um die Lexikoneinträge samt ihrer zugeordneten Konzepte in den natürlichsprachlichen Dokumenten zu finden. Das Verfahren zur Speicherung der Lexika wird in Kapitel 3 näher erläutert. Es ist dazu in der Lage, große Mengen von Wörtern zu speichern und effizient Lexikoneinträge in Texten zu identifizieren.

Die zusätzliche manuelle Arbeit dagegen lässt sich nur bedingt reduzieren, insbesondere wenn man ambige Ausdrücke vermeiden will. Allerdings ist der Aufbau der Lexika, genauso wie der Aufbau der Wissensressource selbst immer auch mit manueller Arbeit verbunden. Dabei können beim Aufbau der Lexikoneinträge schon verschiedenen (semi-) manuelle Techniken zum Einsatz kommen, um verschiedene Flexionsvarianten von Einzelwörtern und Phrasen zu erzeugen.

2.3.4. Zusammenfassung

Anders als die klassischen Verfahren zu Dokumentenindexierung, welche alle *wichtigen* Token der Textdokumente im Index ablegen, geht das Verfahren zur semantischen Indexierung einen anderen Weg. Gegeben eine Menge von eindeutigen abstrakten Konzepten und eine Reihe von natürlichsprachlicher Identifikatoren, die diese referenzieren, werden die Konzepte in den Texten identifiziert und anschließend im Index abgelegt.

Wo das klassische Vorgehen gerade auch bei flektierenden Sprachen wie dem Deutschen danach strebt, die Menge der zu indexierenden Token zu verkleinern, können beim umgekehrten Vorgehen großzügig verschiedene Flektionsformen mit in das Lexikon aufgenommen werden. Die Vergrößerung der Menge der Lexikoneinträge vergrößert hierbei nicht die Menge der eigentlich zu indexierenden Konzepte. Während Stemming und Lemmatisierung eine mehr oder weniger aufwendige, meist sprachabhängige Textvorverarbeitung erfordern, kommt das Verfahren zur semantischen Indexierung mit einer minimalen Textvorverarbeitung aus. Durch den Verzicht von Stemming und Lemmatisierung wird zusätzlich auch das Einführen weiterer Ambiguitäten verhindert.

2.4. Ambiguitäten

Die Problematik von Ambiguitäten in natürlichen Sprachen ist in den vorangegangen Teilen bereits mehrfach erwähnt worden. Wörter natürlicher Sprachen sind meist nicht eindeutig. Betrachtet man bestimmte Wörter losgelöst von ihrem Auftreten im Satzgefüge, ist eine eindeutige Bedeutungszuordnung oft nicht möglich. Erst durch Betrachtung des Weiteren Kontextes wird eine eindeutige Festlegung der Bedeutung möglich. Dies ist ein wichtiges Unterscheidungsmerkmal von natürlichen Sprachen im Gegensatz zu künstlich erzeugten Sprachformalismen, wie z.B. Gesetzestexte oder formale Sprachen — die ja mit dem Ziel erzeugt wurden, mögliche Mehrdeutigkeiten auszuschließen.

Ludwig Wittgenstein malt in seinen *philosophischen Betrachtungen* ein schönes Bild der Sprache, mit allen ihren Eigenheiten:

> Daß die Sprachen (2) und (8) nur aus Befehlen bestehen, laß dich nicht stören. Willst du sagen, sie seien darum nicht vollständig, so frage dich, ob unsere Sprache vollständig ist; — ob sie es war, ehe ihr der chemische Symbolismus und die Infinitesimalnotation einverleibt wurden; denn dies sind, sozusagen, Vorstädte unserer Sprache. (Und mit wieviel Häusern, oder Strassen, fängt eine Stadt an, Stadt zu sein?) Unsere Sprache kann man ansehen als eine alte Stadt: Ein Gewinkel

> von Gässchen und Plätzen, alten und neuen Häusern, und Häusern mit Zubauten aus verschiedenen Zeiten; und dies umgeben von einer Menge neuer Vororte mit geraden und regelmässigen Strassen und mit einförmigen Häusern.[67, Bemerkung 18]

Im Gegensatz zur *Synonymie*, die ausdrückt, dass Wörter mit unterschiedlichen Oberflächenformen eine identische oder zumindest eine sehr ähnliche Semantik besitzen, bezeichnet man mit (semantischer) Ambiguität Wörter, die trotz einer identischen Oberflächenform unterschiedliche semantische Bedeutungen haben (vgl. [51, S. 56]).

Ambiguitäten von Wörtern innerhalb von Sprachen erschweren deren Handhabung ungemein. Sie verhindern eine eindeutige Zuordnung von Wörtern auf ihre Bedeutungen und führen oftmals zu einer Kontextabhängigkeit. Man muss also immer auch die Umgebung des Wortes mit analysieren. Künstliche Sprachen — gemeint sind hier sowohl Sprachen, die zur Verarbeitung von Maschinen erdacht wurden, als auch künstliche Sprachformalismen wie sie in Gesetzestexten und ähnlichem vorkommen — versuchen — auch in Hinblick auf eine einfachere Handhabung — solche Ambiguitäten zu vermeiden. Aber selbst solche künstlichen Sprachen sind nicht gefeit vor Ambiguitäten, die immer wieder für Verwirrung sorgen. Ein schönes Beispiel für solch eine Ambiguität ist die Verwendung des Schlüsselworts `static` in C-artigen Programmiersprachen[3]:

> The keyword `static` (confusingly) means „use internal linkage“. That's an unfortunate leftover from the earliest days of C [55, S. 423].

2.4.1. Wortbasierte Ambiguitäten

Die Gründe für diese Ambiguitäten von Wörtern haben vielerlei Ursachen. In der Linguistik unterscheidet man grundsätzlich zwischen *Polysemie* und *Homonymie*. Polysemie bezeichnet die historische Aufspaltung mehrerer Wortbedeutungen aus einem Wort. Homonymie dagegen beschreibt die Verschmelzung zweier ursprünglich unterschiedlicher Wörter hin zu einem Paradigma. Darüber hinaus wird auch noch zwischen *Homophonen* und *Homographen* unterschieden. Diese bezeichnen Wörter, die entweder auf ähnliche Weise ausgesprochen werden, oder auf gleiche Weise geschrieben werden. Es ist dabei zu beachten, dass Homophone und Homographen wirklich unterschiedliche Phänomene darstellen, auch wenn es hin und wieder zu Überlappungen kommt [36, S. 159].

[3] Interessanterweise kann die Ambiguität von `static` als eine Form von Polysemie — wie im weiteren noch erläutert — angesehen werden: ein historisch noch ähnliches Konzept hat sich über die moderne Prozessorarchitektur in zwei unterschiedliche Konzepte aufgespalten.

Beispiel 2.4.1. Die beiden polysemen Bedeutungen des Worts *Schloss* (*Schließvorrichtung, Gebäude*) haben sich historisch aus einem Morphem entwickelt. Bei dem Wort Kiefer (*der Kiefer, die Kiefer (Baum)*) liegt dagegen Homonymie vor. Es ist durch die Überlappung zweier — historisch noch unterschiedlicher — Worte, mhd. *kiver*, mask. (der Kiefer) und ahd. *kienforha*, fem. (die Kiefer (Baum)), entstanden [51, S. 56]

Die sprachhistorischen Gründe der Ambiguitäten von einzelnen Wörtern sind für die weitere Betrachtung von keiner großen Bedeutung. Wichtig ist vor allem, dass *alle* hier genannten Faktoren für Verarbeitung von Texten eine Rolle spielen. Einzig die Homophone, deren Ambiguität sich nicht in der textuellen Repräsentation widerspiegelt, bilden hierbei eine Ausnahme. Denkt man nun aber an Anwendungen, die auch mit Texten, die mit maschineller Spracherkennung erzeugt wurden, umgehen sollen, wäre zu untersuchen, ob selbst diese Phänomene in solchen Anwendungen von Bedeutung sein könnten, da eben nicht alle Homophone zwangsläufig auch Homographen sind und somit gleich ausgesprochene Wörter nicht immer gleich geschrieben werden müssen.

2.4.2. Namen

Abseits von rein linguistischen Ambiguitäten bieten vor allem die unterschiedliche Ausprägungen von Namen eine Vielzahl endloser Doppeldeutigkeiten. Namen sollen im Allgemeinen der eindeutigen Identifizierung von Individuen dienen. Diese eindeutige Identifizierung gelingt allerdings meist nur in einem beschränkten Rahmen und ist oft ohne weiteren Kontext ebenfalls nicht eindeutig.

Mit dem Begriff *Namen* sind hier nicht nur Namen von Personen gemeint, sondern alle Bezeichner, die bestimmte Individuen und Objekte oder Klassen von bestimmten Objekten bezeichnen. Hierunter fallen nicht nur Namen von geographischen Orten, wie Flüssen oder Städten, sondern auch Namen von Produkten und Organisationen, wie etwa Fahrzeug- oder Markennamen.

Die meisten Namen sind hochgradig ambig. Dabei können Namen sowohl uneindeutig unterschiedliche Objekte oder Objektklassen bezeichnen oder auf der anderen Seite auch mit einfachen Wörtern des Textes verwechselbar sein. Somit weisen Namen — im Gegensatz zu ihrer eigentlichen Intention — keine eindeutigen Zuordnungen auf und müssen, wie die meisten anderen Bezeichnungen auch, auf ihre intendierte Bedeutung hin untersucht werden.

2.4.3. Ambiguitäten in den Lexika von Wissensressourcen

Jenseits der Problematik von Ambiguitäten in der Sprache im Allgemeinen spielen im Speziellen bei der semantischen Indexierung auch beim Aufbau der Lexika Ambiguitäten ein Rolle. Die Lexikoneinträge stellen ja die direkte Verbindung zwischen natürlichsprachlichen Ausdrücken und den eindeutigen Konzepten der Wissensressourcen dar. Jeder Lexikoneintrag ist immer genau einem Konzept der Wissensressource zugeordnet. Dabei sollen die Lexikoneinträge idealerweise möglichst alle Vorkommen der entsprechenden abstrakten Konzepte in Texten erkennen können, ohne sich durch Ambiguitäten in die Irre führen zu lassen.

Bei der praktischen Anwendung von semantischer Indexierung spielen Ambiguitäten eine nicht zu unterschätzende Rolle und müssen schon beim meist manuellen Aufbau der Wissensressourcen beachtet werden. Zuallererst müssen auch die Texte der zu indexierenden Dokumentenkollektion betrachtet werden. Hierbei spielen eine Reihe von Eigenschaften für die möglichen Ambiguitäten eine Rolle:

1. Die Domäne der Texte.
2. Die zeitliche Einordnung der Texte.
3. Die Sprache oder die Sprachen der Texte.
4. Die verwendete Textnormalisierung.

Beispiel 2.4.2. Die Verwendung des Lexikoneintrags *Oder* um das Konzept *Oder(Fluss)* zu referenzieren, führt in deutschen Texten zu vielen falsch-positiven Treffern im Index. Die Menge dieser Fehlerkennungen kann durch eine Textnormalisierung sogar noch vergrößert werden, wenn nicht nur die am Satzanfang vorkommenden sondern alle Vorkommen von *oder* mit in den Index aufgenommen werden. Bei englischen Texten dagegen ist die Verwendung dieses Lexikoneintrags deutlich weniger problematisch.

Zeitliche Faktoren spielen bei diesen Ambiguitäten ebenso eine Rolle. So ist der Lexikoneintrag *Präsident Bush* heutzutage nicht mehr eindeutig, wohingegen der Begriff in 30 Jahre alten Texten noch relativ eindeutig auf ein Konzept verweist.

Die Problematik ambiger Lexikoneinträge tritt sowohl schon bei der Erstellung von Lexika als auch später bei der maschinellen Weiterverarbeitung auf. Grundsätzlich können zwei unterschiedliche Typen von ambigen Lexikoneinträgen identifiziert werden, deren Einordnung aber immer mit Blick auf die 4 oben genannten Eigenschaften der Dokumentenkollektion geschehen muss.

1. *Externe Ambiguitäten*: Lexikoneinträge mit genau einer konzeptuellen Zuordnung innerhalb der Wissensressource.

2. *Interne Ambiguitäten*: Lexikoneinträge mit mehreren möglichen konzeptuellen Zuordnungen innerhalb der Wissensressource.

Externe Ambiguitäten

Externe Ambiguitäten betreffen Lexikoneinträge, die innerhalb der Wissensressource über eindeutige konzeptuelle Zuordnungen verfügen, aber aufgrund ihrer Oberflächenform oder aufgrund der Eingeschränktheit der verwendeten Wissensressource in natürlichsprachlichen Texten nicht eindeutig sind. Im Gegensatz zu internen Ambiguitäten können externe Ambiguitäten nicht einfach in den Lexika identifiziert werden, da hierzu (externes) Wissen außerhalb der Wissensressource nötig ist.

Die Behandlung von externen Ambiguitäten muss meist schon bei der (manuellen) Erzeugung der Lexika geschehen. Hierbei muss zuerst entschieden werden, ob der ambige Begriff überhaupt in das Lexikon aufgenommen werden soll und inwieweit die Ambiguität in Hinblick auf die Eigenschaften der Dokumentenkollektion eine Rolle spielt. Ebenso kann versucht werden, dem Lexikoneintrag bestimmte Kontextwörter hinzuzufügen, die dann dazu beitragen können, diesen dann zu disambiguieren. Als letzter Ausweg kann der Lexikoneintrag im Lexikon als ambig markiert werden. Dies führt dazu, wie später noch genauer erläutert werden wird, dass bei der Indexierung versucht wird, Vorkommen dieses Eintrags in den Texten speziell zu disambiguieren und zu entscheiden, ob der ambige Begriff auf ein Vorkommen des Konzepts hinweist oder nicht.

Beispiel 2.4.3. In einer deutschen Wissensressource ist wiederum der Lexikoneintrag *Oder* ein gutes Beispiel für eine externe Ambiguität. Einerseits weist er in Texten auf das Konzept *Oder(Fluss)* hin, andererseits kann es sich um ein häufig gebrauchtes Funktionswort ohne konzeptuelle Zuordnung handeln.

Dieser Lexikoneintrag kann nun entweder als ambiger Lexikoneintrag markiert werden oder durch Kontextwörter disambiguiert werden. Hierzu könnte man z.B. anstatt des Lexikoneintrags *Oder* die Einträge *die Oder* und *der Oder* hinzufügen.

Interne Ambiguitäten

Interne Ambiguitäten betreffen Lexikoneinträge die innerhalb der Wissensressource mehreren unterschiedlichen Konzepten zugeordnet werden können oder zugeordnet sind[4]. Sie stellen eine Verletzung der in Definition 1.5.3 dargestellten An-

[4]Hierbei sei angemerkt, dass interne Ambiguitäten durchaus auch über weitere externe Ambiguitäten verfügen können, die dann wiederum beiderseits mit den Mitteln der Disambiguierung von externen oder internen Ambiguitäten behandelt werden müssen.

nahme dar, dass es eine Funktion gibt, die Lexikoneinträge eindeutig auf Konzepte abbilden kann und müssen beim Aufbau der Wissensressourcen aufgelöst werden.

Ambige Begriffe können manchmal, ähnlich wie bei externen Ambiguitäten auch, durch Hinzufügen von entsprechenden Kontextwörtern in unterschiedliche Lexikoneinträge mit entsprechenden Zuordnungen aufgespalten werden. Lassen sich keine entsprechend guten Kontexte finden, müssen die ambigen Begriffen entweder aus der Wissensressource entfernt oder mit den Mitteln, wie sie in Kapitel 7 noch beschrieben werden, behandelt werden.

2.4.4. Zusammenfassung

Bei der Behandlung von natürlichsprachlichen Texten und bei der Identifizierung von eindeutigen Konzepten mit Hilfe natürlichsprachlicher Ausdrücke muss das Phänomen von Ambiguitäten beachtet werden und deren Auswirkungen auf die Qualität der Indexierung abgeschätzt werden. Die Klassifizierung und Behandlung von Ambiguitäten ist dabei immer vom Kontext der verwendeten Wissensressource abhängig.

Einerseits muss beim Aufbau der lexikalischen Ressourcen darauf geachtet werden, dass externe Ambiguitäten möglichst nicht in die Lexika aufgenommen werden oder zumindest entsprechend markiert werden, da diese nicht automatisch erkannt werden können. Interne Ambiguitäten dagegen können automatisch erkannt werden und müssen entsprechend behandelt werden. Dies kann auf unterschiedliche Weise geschehen. Andererseits können viele externe und interne Ambiguitäten direkt vermieden werden, indem nicht nur einzelne Wörter, sondern auch Mehrwortlexeme und Phrasen in die Lexika aufgenommen werden.

Der nächste Abschnitt beleuchtet nun die Verwendung von Mehrwortlexemen in den Lexika von Wissensressourcen.

2.5. Mehrwortlexeme

Bei der Behandlung von lexikalischen Ambiguitäten im vorangegangen Abschnitt ist bereits mehrfach angeklungen, dass oftmals ambige Ausdrücke durch das Hinzufügen von Kontextwörtern disambiguiert werden können. Damit kann eine eindeutigere Identifizierung von Konzepten in natürlichsprachlichen Texten erreicht werden, ohne zu viele falsch positive Textstellen zu erkennen. Die Einträge von Lexika umfassen somit nicht mehr nur Einzelwörter sondern im Allgemeinen Phrasen, die aus mehreren Wörtern zusammengesetzte sind.

2.5.1. Arten von Mehrwortverbindungen

Solche komplexen, mehrgliedrigen Satzkonstituenten — in der Literatur meist als *Mehrwortverbindungen*, *mehrgliedrige lexikalische Einheiten* oder *Mehrwortlexeme* bezeichnet — stellen

> […] Verbindungen von zwei oder mehr Lexemen auf syntagmatischer Ebene dar […] [53]

Diese Verbindungen treten in vielen Texten auf. Sie sind oftmals wichtige bedeutungstragende Einheiten im Satzgefüge und stellen wichtige Ankerpunkte zur Behandlung von Texten mit Wissensressourcen dar.

In der linguistischen Literatur werden eine Vielzahl unterschiedlicher Arten von Mehrwortverbindungen unterschieden. Grundsätzlich werden zwischen sogenannten *festen* und *freien* Wortverbindungen unterschieden. Die Unterscheidung basiert dabei auf der Kombinierbarkeit der einzelnen Wortbestandteile der Wortfolgen. Es stellt sich bei der Unterscheidung somit meist die Hauptfrage, ob einzelne Teile des Mehrwortgefüges sinnvoll austauschbar sind. Im Falle von freien Wortverbindungen können einzelne Teile aus dem Gefüge heraugelöst oder entfernt werden. Bei festen Wortverbindungen ist dies dagegen nicht möglich [53].

Da das Hauptaugenmerk im Bezug auf Mehrwortlexeme in dieser Arbeit auf der allgemeinen Behandlung von mehrgliedrigen Ausdrücken liegt und nicht auf der technischen Unterscheidung von komplexen Wortgefügen, wird hier nicht weiter auf die unterschiedlichen Arten von Mehrwortverbindungen eingegangen. Im weiteren Verlauf dieser Arbeit bezeichnen die Begriffe Mehrwortlexem und Mehrwortverbindung alle unterschiedlichen Arten von mehr oder weniger festen Wortverbindungen aus mehreren Token, die einen Lexikoneintrag bilden und bei der weiteren Verarbeitung des Textes als zusammenhängendes Gefüge angesehen werden sollen.

Beispiel 2.5.1. Beispiele für Mehrwortlexeme sind oftmals feste sprachliche Ausdrücke wie *zu tun haben* oder *Ein Mann, ein Wort*, aber auch geographische Bezeichnungen wie *Frankfurt an der Oder* und *New York*. Ebenso sind personenbezogene Konstruktionen wie *Bundeskanzlerin Angela Merkel* oder *Nobelpreisträgerin Marie Curie* wichtige Mehrwortverbindungen.

Im Hinblick auf die Vermeidung ambiger Begriffe spielen aber auch flektierte Phrasen wie *die Oder* bzw. *der Oder* eine Rolle. Solche Phrasen können oftmals dazu eingesetzt werden mit einfachen linguistischen Mitteln bestimmte Formen von Ambiguitäten[5] zu vermeiden.

[5] In diesem Fall die Ambiguität zwischen dem Fluss *Oder* und dem Funktionswort *oder*.

2.5.2. Mehrwortverbindungen in den Lexika

Bei der Behandlung von Mehrwortlexemen durch Lexika muss immer beachtet werden, dass es einen Unterschied macht, ob es sich um eine feste oder freie Mehrwortverbindung handelt. Bei freien Verbindungen besteht — zumindest bei flektierenden Sprachen — die Möglichkeit, dass einzelne Teile der Verbindung in flektierter Form im Text auftreten können. Somit müssen bei der Verwendung von Vollformenlexika bei freien Mehrwortverbindungen auch mögliche flektierte Formen behandelt werden.

Um diesen Mehraufwand an manueller Arbeit zu vermindern, kann beim Eintragen von Lexikoneinträgen auf eine einfache Syntax zurückgegriffen werden, die automatisch verschiedene Formen von Strings generieren kann. Diese Syntax ist an die aus unixoiden Shells bekannte Syntax zur sogenannten *Brace Expansion* [6] angelehnt. Sie kann aus einer Reihe, von geschweiften Klammern *{...}* umschlossenen und durch Kommata getrennten Strings eine Folge von neuen Strings generieren, die aus allen möglichen Verkettungen der Strings bestehen.

Beispiel 2.5.2. So können in etwa die verschiedenen Formen des Paradigmas von *Tisch* durch den Ausdruck *Tisch{,e,es,en,s}* erzeugt werden: *Tisch, Tische, Tisches, Tischen, Tischs.*

Auch können systematisch verschiedene Strings zur Identifizierung verschiedener Datumsangaben durch den Ausdruck *{03,3}{,.}{ ,}{09,9,September,Sep}{,.}{1983}* erzeugt werden.

Die Leerzeichen und das leere Wort sind in diesem Ausdrücken explizit zu verstehen: leere Wortfolgen geben immer die Verkettung mit dem leeren Wort ε an und ein explizites Leerzeichen die Verkettung mit eben diesem.

Komplexe Mehrwortverbindungen in Vollformenlexika erzeugen ihrerseits wieder verschiedene Herausforderungen. Einerseits muss es möglich sein, deutlich längere Stringverbindungen abzuspeichern, andererseits müssen die Lexika weiterhin effizient nach Einträgen durchsuchbar bleiben. Auch sollen die Mehrwortverbindungen auch über Zeilenumbrüche und verschiedenartigen Leerzeichen hinaus in den Texten der Dokumentenkollektion identifiziert werden.

Bei der Verwendung von Lexika zur Handhabung von Mehrwortlexemen stellen sich somit mannigfaltige Fragen nach der genauen Verwendung und nach einer effizienten Speicherung und Handhabung solcher großer und komplexer Datensammlungen. Auch ist das Auffinden von komplexen Wortverbindungen aufwendiger, da einfaches token-basiertes Nachschlagen in Wörterbüchern nicht mehr so ohne weiteres möglich ist. Diese Problematik und ein Ansatz zur Lösung soll in späteren Kapiteln noch weiter behandelt werden.

2.6. Historische Schreibvarianten und Fehler

Einen weiteren Aspekt bei der Behandlung von natürlichsprachlichen Texten stellen zum einen verschiedenartige (historische) Schreibvarianten von Wörtern und zum anderen Fehler aus unterschiedlichsten Quellen dar. Die Konsequenzen für unterschiedliche Anwendungen sind dabei stark von dem bearbeiteten Thema abhängig. Während bei der Behandlung moderner Texte das Phänomen von vereinzelten Rechtschreibfehlern oft vernachlässigbar ist, können dagegen Anwendungen, die sich mit historischen Dokumenten aus automatischer Zeichenerkennung beschäftigen, diese Phänomene nicht einfach ignorieren. Gerade für die letztgenannten Anwendungen stellt die Vermischung von Erkennungsfehlern mit historischen Rechtschreibvarianten und uneinheitlichen Schreibweisen in Texten eine nicht zu unterschätzende Herausforderung für Anwendungen dar [20, 47].

Uneinheitliche Rechtschreibung ist allerdings nicht nur ein Phänomen historischer Textdokumente. Auch moderne Texte können durchaus unterschiedliche Schreibweisen für einzelne Wörter aufweisen. Oftmals handelt es sich um Schreibvarianten unterschiedlicher Dialekte, wie z.B. dem amerikanischen und britischen Englisch oder dem Deutschen und Österreichischen. Gerade auch im Deutschen sind durch die jüngste Rechtschreibreform des Deutschen im Jahr 1996 verschiedenen Schreibvarianten entstanden. Ein weitere Quelle für inkonsistente Schreibweisen sind verschiedene Marken-, Artikel- und Projektnamen. Solche Namen werden oft auf verschiedene Arten, mit oder ohne Bindestrichen und mit verschiedenartiger Groß- und Kleinschreibung ausgeschrieben.

Im weiteren Verlauf werden historische Schreibvarianten und Fehler gleich behandelt. Für die folgenden Betrachtungen spielt es keine Rolle, wie genau ein Fehler zustande gekommen ist. Es muss dabei jedoch angemerkt werden, dass gerade historische Schreibvarianten[6] immer auch durch entsprechende Erweiterungen im Lexikon behandelt werden können. So können zum Beispiel die verwendeten natürlichsprachlichen Lexika Einträge in neuer und alter Rechtschreibung oder in moderner und historischer Schreibweise enthalten. Ebenso können moderne Lexikoneinträge durch die Anwendung von historischen Rewrite-Pattern automatisch historisiert werden [15, 17].

Die Anwendung unterstützt solche Formen von Erweiterungen. Es können dabei ähnliche Techniken angewendet werden, wie bei der Erzeugung unterschiedlicher, flektierter Lexikoneinträge (vgl. Beispiel 2.5.2), indem die Lexikoneinträge entweder manuell oder auch automatisch mit entsprechenden Pattern erweitert werden.

[6] aber auch bestimmte Formen von Tippfehlern, OCR-Erkennungsfehlern usw.

Beispiel 2.6.1. Beispiele für historisch bedingte Schreibvarianten sind unter anderem Übergänge wie *Theyl* → *Teil* und *daß* → *dass*. Fehler von Menschen und auch von Maschinen können ähnlich aufgefasst werden. Sie umfassen Fehler der automatischen Zeichenerkennung wie *Stiick* → *Stück* und auch menschliche Fehler wie etwa Vertauschungen einzelner Buchstaben (*rigth* → *right*).

In den Lexika der Wissensressourcen können dann einfach entsprechende erweiterte Lexikoneinträge verwendet werden: *{Th,T}e{i,y}l*, *da{ss,ß}* und *rig{ht,th}*.

2.6.1. Eigenschaften von Fehlern

Für die weiteren Betrachtungen ist es also unerheblich, um welche Art von Fehlern es sich handelt. In diesem Kontext ist allen Fehlern gemein, dass sie allesamt fehlerhafte Zeichenfolgen darstellen, deren eigentliche Bedeutung im Textgefüge rekonstruiert werden soll. Das Auftreten von Fehlern folgt meist einer bestimmten Systematik auf Zeichenebene. Diese *Fehlermuster* umfassen folgende Phänomene:

- Vertauschungen der Reihenfolge mehrerer aufeinanderfolgender Zeichen.
- Vertauschungen einzelner Zeichen durch andere.
- Auslassungen einzelner Zeichen.
- Einfügungen einzelner Zeichen.
- Vereinigung mehrerer Zeichen zu einem.
- Aufspaltung einzelner Zeichen zu mehreren.

Dabei wird die Anzahl dieser Fehlermuster meist noch weiter reduziert, da einzelne separat genannte Muster auf eine Folge *atomarer Editieroperationen* zurückgeführt werden können. Die atomaren Editieroperationen umfassen zumindest *Insertions*, *Deletions* und *Substitutions*. Des Weiteren werden oft noch *Splits* und *Merges* verschiedener Länge betrachtet.

Natürlich treten einzelne dieser Fehlermuster gehäuft bei bestimmten Fehlerquellen auf, wohingegen andere Fehlermuster in bestimmten Fehlerquellen so gut wie nie auftreten. Diese Unterschiede in den Verteilungen solcher Fehler können später dann bei der Behandlung von Fehlern ausgenutzt werden, um bessere *Korrekturkandidaten* zur Fehlerbehebung zu erhalten. Die Menge der Korrekturkandidaten bezeichnen hier immer eine Menge von potentiellen Verbesserungsvorschlägen für ein fehlerhaftes Wort oder eine fehlerhafte Mehrwortverbindung.

Fehlermuster treten nicht nur einmal pro Wort auf. Es ist durchaus möglich, dass sich verschiedene Fehlermuster sowohl aus unterschiedlichen Quellen als auch

Abbildung 2.1.: Fehlermuster in historischen Dokumenten (frei nach [47]).

aus den selben Quellen überlagern und insbesondere auch mehrfach in einzelnen Token und Mehrwortverbindungen auftauchen.

Beispiel 2.6.2. Gerade bei der Arbeit mit historischen Texten treten verschiedene Ausprägungen gemeinsam auf. Abbildung 2.1 zeigt ein Beispiel für die Arbeit mit historischen Textdokumenten und die Vielzahl von Fehlern, die dabei auftreten können. Sie verdeutlicht insbesondere die Überlappung von historischen Schreibvarianten (*Franckreich* → *Frankreich*) mit fehlerhaft erkanntem Text aus Bilddokumenten (*Zranckreich* → *Franckreich*).

Die Abbildung zeigt am Beispiel *Kö « nigen* → *Königen* auch eine fehlerhafte Tokenisierung, wie sie oft bei der automatischen Zeichenerkennung von historischen Dokumenten auftritt.

Die einzelnen Zeichen, die in solchen Fehlermustern auftreten, umfassen nicht nur die „normalen" Buchstaben einer Sprachen sondern verschiedene Formen weiterer Zeichen, wie unter anderem verschiedene Arten von Leer- und Interpunktionszeichen (vgl. dazu Abbildung 2.1 und Beispiel 2.6.2). Dies kann unter bestimmten Umständen auch zu fehlerhafter Tokenisierung führen und liefert ein weiteres Argument für die stabile Handhabung von komplexen Mehrwortverbindungen, wie sie bereits im vorhergegangenen Kapitel 2.5 dargelegt wurde.

2.6.2. Handhabung von Fehlern

Die Handhabung von Fehlern hängt stark von der Anwendung und insbesondere von den Texten ab, die bearbeitet werden sollen. Einen wichtigen Faktor stellen vor allem die Quellen, aus denen die Textdokumente stammen, dar. Hat man es insbesondere mit analogen Textquellen zu tun, ist die Handhabung von Fehlern von zentralerer Bedeutung als bei Texten aus digitalen Quellen. Natürlich spielen bei analogen Textquellen vor allem die Qualität der Ursprungsdokumente und die Qualität der maschinellen Erzeugung von Texten eine tragende Rolle. Des Weiteren sind die Sprache der Texte und — wichtiger noch — die Zeitstufe der Texte weitere Faktoren, die bei der Behandlung von Fehlern eine wichtige Rolle spielen.

Hat man es insbesondere mit modernen Texten aus vornehmlich digitalen Quellen zu tun oder ist aufgrund anderer Umstände mit wenigen Fehlern in den Tex-

ten zu rechnen, kann oftmals gänzlich auf eine weitere Behandlung von Fehlern verzichtet werden. Eine naheliegende Erweiterung, um einfache Schreibvarianten von Wörtern zu handhaben, können diese Schreibvarianten zusätzlich direkt in bereits vorhandene Lexika eingebunden werden. Hierzu können verschiedene Muster auch automatisch auf die Lexikoneinträge angewendet werden, um entsprechende Schreibvarianten zu erzeugen [16, 18]. Diese Technik — häufige Schreibvarianten direkt in bestehende Lexika aufzunehmen — kann im weiteren dabei helfen bestimmte Arten von Schreibvarianten ohne weiteren Aufwand direkt in den Texten zu finden.

Beispiel 2.6.3. Will man die semantische Indexierung auf Texten anwendenden, die aus irgendwelchen Gründen das lange S „*ſ*“ enthalten, kann zum Beispiel einfach für jeden Lexikoneintrag automatisch eine Variante erzeugt werden, indem die Vorkommen von *s*, *ss* und *ß* durch entsprechende Folgen von *ſ* ersetzt werden. So würde zusätzlich zu dem Lexikoneintrag *„Deutschland“* ein weiterer Lexikoneintrag *„Deutſchland“* mit der gleichen konzeptuellen Zuordnung erzeugt werden.

Ist dagegen mit vielen Fehlern in den Texten zu rechnen, können sich die auftretenden Fehler durchaus negativ auf die semantische Indexierung auswirken, da unter Umständen zu wenig Lexikoneinträge — und damit zu wenig Konzepte — in den einzelnen Dokumenten identifiziert werden können. Um in solchen Fällen die Konzepte besser in den Dokumenten identifizieren zu können, können bei der semantischen Indexierung nach einer ersten, exakten Suche mit einem Lexikon auch mehrere, unscharfe Suchen nachgeschaltet werden. Hierzu werden die Wörter bzw. die Mehrwortverbindungen der Eingabedokumente nach Lexikoneinträgen durchsucht, zu denen sie einen, zuvor festgelegten, maximalen Levenshteinabstand (vgl. Kapitel 1.4) nicht überschreiten. Da die Lexika komplexe Mehrwortverbindungen enthalten können und auch die möglichen Fehler die Tokenisierung der Eingabedokumente beeinflussen können (vgl. Abbildung 2.1), muss die vorgenommene unscharfe Suche solche Phänomene explizit handhaben können.

Im allgemeinen Fall — vor allem bei kurzen Wörtern oder zu großen maximalen Levenshteinabständen — können einem bestimmten Wort bzw. einer bestimmten Mehrwortverbindung im Eingabedokument mehrere Lexikoneinträge zugeordnet werden, die wiederum verschiedenen Konzepten zugeordnet sind[7]. Durch die unscharfe Suche können somit weitere Ambiguitäten eingeführt werden, die wiederum bei der semantischen Indexierung behandelt werden müssen.

[7] Sind dagegen mehrere mögliche Lexikoneinträge alle demselben Konzept zugeordnet, tritt kein Problem auf, da zwar der genaue Lexikoneintrag nicht bekannt ist, sehr wohl aber das genaue Konzept.

2.7. Textnormalisierung

Der letzte hier behandelte Aspekt ist eher computertechnischer als linguistischer Natur. Die *Textnormalisierung* bzw. *Normalisierung* umfasst eine Reihe von Techniken, die von verschiedenen Anwendungen bei der Textverarbeitung in sehr frühen Schritten eingesetzt wird. Der Einfluss der Normalisierung von Texten auf die hier gegebenen Betrachtungen soll hier nur kurz angeschnitten werden. Die Normalisierung umfasst einfache Techniken wie die Anpassung der Groß- und Kleinschreibung eines Textes und verschiedene Säuberungstechniken von unterschiedlichen Interpunktionszeichen. Sie kann aber auch komplexere Techniken wie die Stammformenreduktion oder die Lemmatisierung umfassen.

Ein einfacher Vorteil der Normalisierung ist die Reduktion der zu behandelnden Zeichen in den Lexikoneinträgen der expliziten Wissensressourcen. So kann vor allem auch die Größe der benötigten Lexika deutlich reduziert werden. Auf der anderen Seite kann die Normalisierung unter Umständen auch eine weitere Quelle für unnötige Ambiguitäten sein.

Bei der Textnormalisierung ist vor allem wichtig, dass sie immer sowohl auf die Eingabedokumente als auch auf die Lexikoneinträge der verwendeten Lexika angewendet werden muss, da es ansonsten zu Inkonsistenzen zwischen den Textdokumenten und den Lexika kommen kann. Dies kann dann dazu führen, dass verschiedene Lexikoneinträge nicht mehr in den Eingabedokumenten gefunden werden können.

Die hier dargestellte semantische Indexierung verwendet nur eine rudimentäre, optionale Textnormalisierung, die Großbuchstaben zu Kleinbuchstaben umwandelt und Punktuationszeichen aus den Texten entfernt. Für erweiterte Anwendungen können bei der Textnormalisierung auch verschiedene andere Techniken — wie etwa Lemmatisierung oder Stemming — zur Anwendung kommen.

2.8. Zusammenfassung

In diesem Kapitel wurden eine Reihe von linguistischen Problemen bei der Arbeit mit natürlichsprachlichen Texten erörtert und eine kurze Übersicht über Techniken gegeben, die dabei helfen können, die jeweiligen Probleme zu lösen. Dabei ist auch klar geworden, dass der Einfluss der einzelnen Probleme stark auf der Textart beruht, die von einer Anwendung verarbeitet werden soll. Anwendungen, die flexibel mit einer Vielzahl von unterschiedlichen Texten aus verschiedenen Quellen umgehen sollen, müssen somit verschiedene Techniken dynamisch miteinander kombinieren können, um Texte unterschiedlicher Art verarbeiten zu können. Dies ist notwendig, um die spätere Bearbeitung mit Wissensressourcen zu ermöglichen

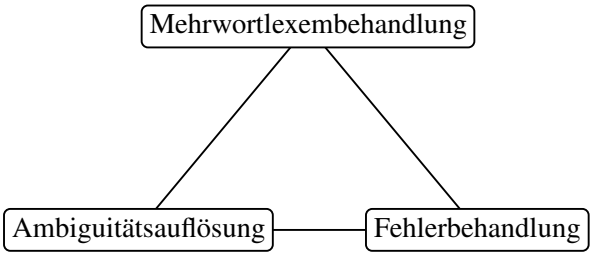


Abbildung 2.2.: Abhängigkeiten bei der Handhabung natürlichsprachlicher Texte

beziehungsweise zu verbessern. Die in diesem Kapitel erörterten Aspekte können meist nicht einfach einzeln, sondern müssen immer zusammenhängend betrachtet werden.

Die Handhabung von Mehrwortverbindungen, wie im Kapitel 2.5 ab Seite 36 erläutert, stellt die Basis einer stabilen Textverarbeitung dar. Einerseits kann eine explizite Handhabung solcher Tokenverbindungen das Auftreten vieler wortbasierter Ambiguitäten von vornherein verhindern und hat somit einen positiven Einfluss auf die gesamte Problematik der semantischen Disambiguierung. Die Ambiguitätsauflösung erfordert somit auch eine explizite Handhabung von Mehrwortlexemen in den Lexika. Andererseits müssen nun bei der Fehlerbehandlung auch explizit Mehrwortverbindungen mit beachtet werden.

Bei der Fehlerkorrektur sollten auch Fehler in der Tokenisierung behandelt werden können. Somit ist auch bei der Fehlerkorrektur die explizite Behandlung von Mehrwortverbindungen wichtig. Wie bereits erörtert, kann die Behandlung von Fehlern weitere Ambiguitäten einführen, die bei der Fehlerkorrektur meist noch nicht eindeutig aufgelöst werden können. Hier hat die Fehlerkorrektur einen direkten Einfluss auf die Weiterverarbeitung des Textes und es wird eine Form von Ambiguitätsauflösung notwendig.

Abbildung 2.2 verdeutlicht die Abhängigkeiten der in diesem Kapitel erörterten Aspekte. Die zirkulären Abhängigkeiten der verschiedenen Aspekte erfordern somit immer eine ganzheitliche Handhabung aller Aspekte.

Zur Behandlung der einzelnen Aspekte wurden immer auch Lösungsansätze auf der Basis der Lexika erörtert. Alle Aspekte benötigen unterschiedliche Formen von (Vollformen-) Lexika zur Auflösung. Es ist klar, dass zu der hier genannten Behandlung von Texten, einfache Vollformenlexika auf der Basis von einzelnen Token nicht mehr ausreichen. Vielmehr werden Lexika benötigt, die eine große Zahl komplexer Tokenverbindungen und derer Flexionsformen abbilden. In die-

sen Lexika müssen sich natürlich auch immer die verwendeten Normalisierungstechniken widerspiegeln. Das soll heißen, dass sich die Normalisierungstechniken immer auch nach den vorhandenen Lexika richten müssen.

Der erste Schritt bei der Arbeit mit Texten ist dann immer die Suche nach den Lexikoneinträgen in den Texten. Auf Basis dieser Lexikoneinträge und der nicht erkannten Tokenverbindungen kann dann eine weitere Fehlerbehandlung und Ambiguitätsauflösung stattfinden. Ein erster einfacher Ansatz zur Textbehandlung mit einer Wissensressource wird nun im folgenden Kapitel dargestellt. Dieser Ansatz soll dann im Folgenden noch weiter verfeinert werden.

Auf die Behandlung von Fehlern und historischen Schreibvarianten bei der semantischen Indexierung mit expliziten Wissensressourcen wird in Kapitel 6 eingegangen, auf die Behandlung von Ambiguitäten in den Wissensressourcen und in der Dokumentenkollektion in Kapitel 7. Der Aufbau geeigneter Vollformenlexika mit Mehrwortlexemen stellt dagegen keinen Gegenstand dieser Arbeit dar und die in diesem Kapitel dargestellten Überlegungen zu den Lexika von expliziten Wissensressourcen — gerade auch in Hinblick auf mögliche externe und interne Ambiguitäten — sollen nur eine grobe Übersicht über diese Problematik geben.

3. Computerlinguistische Grundlagen zum effizienten Zugriff auf Wissensressourcen

Zur semantischen Indexierung von Dokumenten müssen wichtige komplexe Schlüsselbegriffe in natürlichsprachlichen Texten identifiziert und den Konzepten der Wissensressource zugeordnet werden. Hierzu werden effiziente Verfahren benötigt, die eine große Anzahl mehrgliedriger Lexikoneinträge verwalten und diese in den Dokumenten finden können. In diesem Kapitel werden automatenbasierte Verfahren zur Repräsentation der Lexika vorgestellt, die sowohl die Speicherung beliebig langer Lexikoneinträge, als auch eine effiziente Suche in Texten unterstützen und auch die Zuordnung zwischen Lexikoneinträgen und abstrakten Konzepten der Wissensressourcen zulassen.

3.1. Das Lexikon der semantischen Indexierung

Bei der semantischen Indexierung von Textdokumenten werden zwei konzeptionell voneinander getrennte externe Ressourcen verwendet. Die Informationen über die Konzepte und deren semantische Verbindungen untereinander ist in den expliziten Wissensressourcen abgespeichert. Die eindeutigen natürlichsprachlichen Ausdrücke der Konzepte sind dagegen in den Lexika abgespeichert, wobei die einzelnen Lexikoneinträge immer eindeutig einem Konzept der Wissensressource zugeordnet sind (siehe unten).

Wie im vorangegangenen Kapitel erläutert, sind in den Lexika nicht nur einzelne Wörter abgespeichert. Vielmehr sollen aus verschiedenen Gründen auch flektierte Mehrwortverbindungen in den Lexika abgespeichert werden können. Die Lexika müssen somit in der Lage sein diese umfangreichen Mengen von Zeichenketten effizient abzuspeichern und schnell auffindbar zu machen. Darüber hinaus müssen einzelne Lexikoneinträge eindeutig ihren Konzepten der Wissensressource zugeordnet werden können und direkt beim Auffinden eines Lexikoneintrags verfügbar sein.

Einfache, textbasierte Aufgaben verwenden oft einfache Datenstrukturen wie sortierte Listen, binäre Suchbäume oder verschiedene Hashtabellen zur Repräsentation von Lexika. Diese Datenstrukturen bieten sowohl eine einfache Möglichkeit der Zuordnung zwischen Einträgen und ihren Konzepten als auch ein schnelles Nachschlagen von Einträgen. Leider eigenen sich diese Datenstrukturen nur bedingt zur Handhabung von Mehrwortverbindungen, da sie nur ein Nachschlagen nach vollständigen Einträgen erlauben und kein Möglichkeit bieten, inkrementell und buchstabenweise vorzugehen.

Bei der Verwendung solcher Datenstrukturen müssten Mehrwortverbindungen in den Texten somit schon vor der eigentlichen Suche nach Lexikoneinträgen als solche identifiziert worden sein und so sind weitere Datenstrukturen oder andere algorithmisches Techniken nötig um Mehrwortverbindungen in den natürlichsprachlichen Texten zu handhaben. Ein weiterer Nachteil dieser Strukturen ist, dass sie die Einträge in ihrer Gesamtheit abspeichern. Sie sind meist nur schlecht in der Lage, gemeinsame Prä- und Suffixe der Einträgen zu identifizieren und effizient gemeinsam abzuspeichern.

Aus diesen Gründen sind für die semantische Indexierung spezielle Techniken zur Speicherung von großen Lexika besser geeignet. Diese Techniken sind zwar nicht neu, dennoch sind sie nicht allgemein bekannt und auch nicht besonders weit verbreitet. Im Rahmen dieser Arbeit wurden eigene Implementierung der Grundverfahren zur Lexikonrepräsentation entwickelt und die Verfahren auf die speziellen Anforderungen der semantischen Indexierung angepasst.

3.2. Konzeptuelle Zuordnungen

Die Sprachdaten, die bei der semantischen Indexierung zur Verwendung kommen, müssen in aller erste Linie natürlichsprachliche Ausdrücke mit ihren abstrakten Konzepten in der Wissensressource in Verbindung bringen. Hierzu müssen die verwendeten Implementierungen nicht nur in der Lage sein zu entscheiden, ob eine bestimmte Folge $w \in \Sigma^*$ eines Alphabets Σ ein Lexikoneintrag ist oder. Vielmehr muss es eine geeignete Implementierung zur Repräsentation von Lexika ermöglichen, für jeden Lexikoneintrag $w \in \mathcal{W}_{Lex}$ der Sprache des Lexikons einer Wissensressource $\mathcal{W}$ dessen *konzeptuelle Zuordnungen* κ zu bestimmen. Die konzeptuellen Zuordnungen des Lexikons $\kappa : \mathcal{W}_{Lex} \rightarrow \mathcal{W}_K$ weisen jedem Wort des Lexikons genau ein eindeutiges Konzept der Konzeptmenge $\mathcal{W}_K$ der Wissensressource zu (vgl. Definition 1.5.1).

Die konzeptuellen Zuordnungen κ bilden eine totale Abbildung. Das heißt, dass immer jedem Wort des Lexikons $\mathcal{W}_{Lex}$ eindeutig genau ein Konzept $k \in \mathcal{W}_K$ zugeordnet ist. Diese Zuordnungen sind im Allgemeinen weder *injektiv* noch *surjektiv*

— mehrere, verschiedene Lexikoneinträge können jeweils auf das gleiche Konzept verweisen und nicht jedes Konzept der Konzeptmenge muss von einem Lexikoneintrag referenziert werden.

Dies bedeutet insbesondere auch, dass einige Konzepte der Konzeptmenge $k \in \mathcal{W}_K$ von mehreren Lexikoneinträgen referenziert werden können. Tatsächlich existieren für die meisten abstrakten Konzepte mehrere natürlichsprachliche Ausdrücke. Meist enthält die Konzeptmenge nur eine kleine Anzahl von Konzepten, die von keinem natürlichsprachlichen Ausdruck referenziert werden. Solche Konzepte erfüllen meist funktionale Aufgaben innerhalb der Hierarchie der Konzepte und sollen meist nicht direkt in natürlichsprachlichen Dokumenten identifiziert werden[1].

In der Praxis werden die konzeptuellen Zuordnungen direkt mittels Pointer auf die entsprechenden Konzepte oder über eindeutige Identifikationsnummern implementiert. Bei den folgenden Betrachtungen sind die konzeptuellen Zuordnungen einfache Identifikationsnummern, die an den Finalzuständen eines Automaten abgespeichert sind. Eine Identifikationsnummer gleich 0 deutet an, dass ein für ein gegebenes Wort keine konzeptuelle Zuordnung existiert und dieses Wort daher nicht Teil des Lexikons ist und damit über keine konzeptuelle Zuordnung innerhalb der Wissensressource verfügt.

3.3. Tries

Um die Eingabetexte effizient verarbeiten zu können, kann man Lexika als sogenannte *Tries* repräsentieren. Tries oder auch *Präfixbäume* sind Suchbäume die die gemeinsamen Präfixe ihrer Einträge jeweils nur einmal abspeichern. Sie können als spezielle Formen deterministischer Automaten aufgefasst werden, die alle Wörter akzeptieren, die Teil der Sprache des repräsentierten Lexikon ist. Wie andere endliche Automaten auch unterstützen sie eine inkrementelle buchstabenweise Suche und eignen sich somit auch zur Behandlung von Mehrwortverbindungen in Texten.

Da Tries nur die gemeinsamen Präfixe und nicht die gemeinsamen Suffixe der Lexikoneinträge unifizieren, bleibt eine eindeutige Zuordnung zwischen einem Finalzustand des Tries und dem erkannten Wort bestehen. Ein als Trie repräsentiertes Lexikon hat also immer genau so viele Finalzustände wie es Einträge hat. So können an den Finalzuständen des Tries die notwendigen konzeptuellen Zuordnungen der entsprechenden Einträgen abgelegt werden und die nötigen Verbindungen zwischen lexikalischen Einträgen und Konzepten erreicht werden.

[1] Beispiele hierzu sind unter anderem künstlich eingefügte Konzepte zu ambigen Lexikoneinträgen (vgl. Abbildung 7.2) oder die Wurzelknoten von EFGT-Netzen (vgl. Abbildung 4.2).

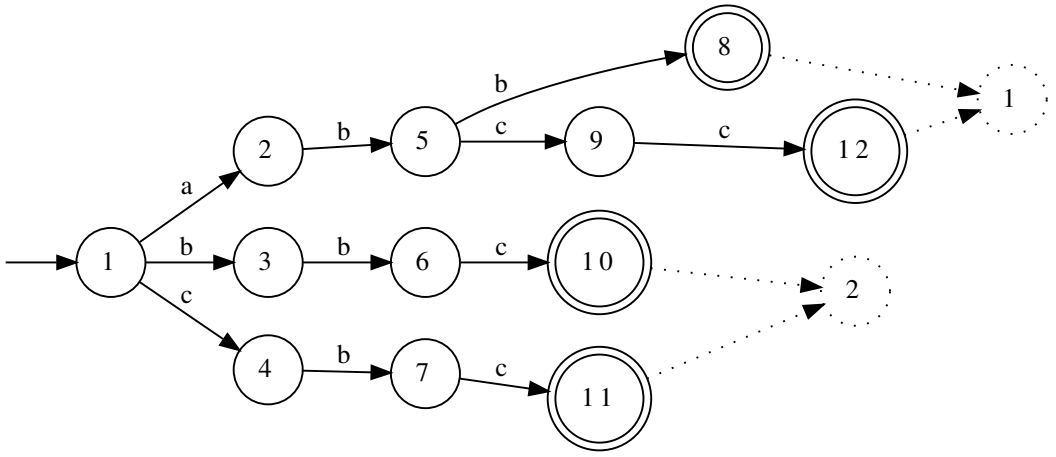


Abbildung 3.1.: Einfacher Trie der Sprache $\mathcal{L} = \{abc, abbc, bbc, cbc\}$ mit den konzeptuellen Zuordnungen $\iota = \{(abc, 1), (abbc, 1), (bbc, 2), (cbc, 2)\}$.

Abbildung 3.1 zeigt ein einfaches als Trie realisiertes Lexikon. Die konzeptuellen Zuordnungen der Lexikoneinträge sind durch die gepunkteten Pfeile und Kreise markiert. Der einzige Startzustand des Automaten ist Zustand 1. Dies wird durch den aus dem Nichts kommenden Pfeil angedeutet. Einer gängigen Notation folgend, werden Finalzustände durch eine doppelte Umrandung markiert. Der dargestellte Automat unifiziert alle gemeinsamen Präfixe der Einträge, während die gemeinsamen Suffixe dagegen mehrfach abgespeichert werden, um die Eindeutigkeit der Zuordnungen zu erhalten. Die Anzahl der Finalzustände des dargestellten Tries entspricht dabei genau der Anzahl der Einträge im Trie. Jeder Finalzustand des Tries hat immer *genau eine* konzeptuelle Zuordnung ungleich 0, wobei mehrere verschiedene Finalzustände des Tries dieselbe konzeptuelle Zuordnung aufweisen können.

3.3.1. Speicherung von Tries in Übergangstabellen

Eine einfache Form, um deterministische Automaten im Allgemeinen und Tries im Speziellen darzustellen, sind sogenannte *Übergangstabellen*. Diese Übergangstabellen definieren für jeden Zustand des Automaten und für jeden Buchstaben des Alphabets den Übergang von einem Zustand mit einem Buchstaben in einen anderen.

Tabelle 3.1 zeigt eine Übergangstabelle für den in Abbildung 3.1 dargestellten

Zustand	1	2	3	4	5	6	7	8	9	10	11	12
a	2	0	0	0	0	0	0	0	0	0	0	0
b	3	5	6	7	8	0	0	0	0	0	0	0
c	4	0	0	0	9	10	11	0	12	0	0	0

Tabelle 3.1.: Die Übergangstabelle eines Tries mit der Sprache $\mathcal{L} = \{abc, abbc, bbc, cbc\}$ (vgl. Abbildung 3.1)

Trie[2]. Jede Zelle der Tabelle definiert den Zielzustand, der erreicht wird, falls man von einem Zustand, welcher durch die Spalte der Tabelle definiert ist, mit einem bestimmten Buchstaben, welcher durch die Zeile der Tabelle definiert ist, dem Übergang folgt. So landet man von Startzustand 1 beim Lesen des Zeichens *b* im Zustand 3. Die Zustandsnummer 0 bezeichnet in dieser Tabelle, dass kein Übergang von einem Zustand aus mit dem entsprechenden Buchstaben möglich ist. Man kann sich diese Zustandsnummer als einen speziellen, nicht finalen Fallenzustand des Automaten vorstellen, von dem aus man mit jedem Buchstaben wiederum in diesem landet (vgl. Lemma 1.3.1). Zusätzlich zu der Übergangstabelle muss auch noch der Startzustand des Tries (in diesem Falle Zustand 1) abgespeichert werden.

Übergangstabellen sind eine einfache und effiziente Art Automaten zu simulieren. Sogar in Tabelle 3.1 des relativ kleinen Tries aus Abbildung 3.1 sind fast 70 Prozent der Zellen mit einer redundanten 0 belegt, die lediglich anzeigt, dass es in der jeweiligen Konstellation aus Zustandsnummer und Buchstaben keinen Übergang gibt.

Die Anzahl der Einträge in einer solchen Tabelle entspricht somit der Anzahl der Zustände mal der Anzahl der Buchstaben des Alphabets. Bei Tries mit vielen Zuständen aus einem großen Alphabet kann die Größe der Tabelle sehr leicht so stark anschwellen, dass sie nicht mehr sinnvoll zu benutzen ist und die Vorteile der Übergangstabelle durch die schiere Größe aufgehoben werden. Besonders ungünstig ist dabei, dass in großen Übergangstabellen vor allem auch nicht existente Zustandsübergänge explizit abgespeichert werden müssen. Dabei haben mit normalen Übergangstabellen repräsentierte Tries oft die ungünstige Eigenschaft, dass besonders viele unnötige Einträge verwaltet werden müssen. Dies liegt darin begründet, dass nur die Zustände weiter links im Trie sehr stark verästeln und viele Übergänge mit vielen unterschiedlichen Buchstaben aufweisen. Von diesen Zuständen gehen viele Übergänge aus und es muss nur wenig redundante Infor-

[2] In der dargestellten Tabelle fehlen zur besseren Übersicht sowohl die konzeptuellen Zuordnungen als auch die Informationen über die Final- und Startzustände des Automaten. Diese Informationen können jedoch einfach durch weitere Zeilen bzw. Spalten an die Tabelle angefügt werden.

mation abgespeichert werden. Je tiefer sich ein Zustand allerdings im Trie befindet desto weniger verzweigt ist er typischerweise und desto weniger Übergänge gibt es für einen Zustand. So muss gerade für diese häufig im Trie auftretenden Zustände eine große Menge redundanter Information gespeichert werden.

3.3.2. Speicherung von Tries mittels verschränkter Tabellen

Zur Verwaltung großer, als Trie repräsentierter Lexika braucht man aus den eben genannten Gründen somit eine bessere Methode, um alle unterschiedlichen Zustände effizient und platzsparend abzuspeichern. Man möchte dabei vor allem verhindern, dass die redundanten Informationen über nicht existente Übergänge im Trie nicht explizit abgespeichert werden müssen. Auf diesem Wege sollten viele explizit zu speichernde Informationen wegfallen und es kann so eine Menge Platz eingespart werden.

Ein Algorithmus, um solche spärlich besetzten Tabellen mit vielen redundanten Informationen platzsparender abzuspeichern, wird in [56] vorgestellt. Die grundlegende Idee hinter dem Algorithmus stellt eine Verschränkung bzw. Verzahnung der Zeilenvektoren einer Tabelle ineinander dar. Dabei wird — vereinfacht ausgedrückt — versucht, diejenigen Zellen von Zeilenvektoren mit solchen Zellen anderer Zeilenvektoren zu überdecken, so dass jeweils nur redundante, nicht weiter benötigte Informationen überschrieben werden. Abbildung 3.2 zeigt das grundlegende Verfahren der Verschränkung von Zeilenvektoren.

Das Verfahren der Verschränkung kann einfach modifiziert werden und so auf die Übergangstabellen von Tries angewandt werden. Da die Übergangstabellen von Tries sehr groß werden können, muss der Aufbau der Übergangstabelle verhindert werden. So kann der Algorithmus nicht einfach auf die Übergangstabelle des Tries angewendet werden. Vielmehr muss der Algorithmus die verschränkte Tabelle direkt aufbauen, ohne vorher den Umweg über die Übergangstabelle zu gehen.

Der angepasste Algorithmus baut die verschränkte Tabelle nun inkrementell Wort für Wort auf. Hierzu muss beim Aufbau des Tries sichergestellt werden, dass immer nur solche Zustände in die Tabelle eingefügt werden, die sich nicht mehr ändern können. Das heißt insbesondere, dass ein Zustand des Tries erst dann in die Tabelle eingefügt werden darf, wenn sicher ist, dass egal welche Lexikoneinträge in Zukunft noch eingefügt werden, dieser Zustand und alle Zustände weiter rechts keine Änderungen mehr benötigen. Um eben dies sicherzustellen, muss der Aufbau der Tabelle des Tries in lexikographischer Reihenfolge der Einträge erfolgen. So können immer all jene Zustände des Vorgängerworts abgespeichert werden, die hinter dem längsten, gemeinsamen Präfix der beiden Wörter liegen. Diese Zustände sind aufgrund der lexikographischen Einfügereihenfolge fixiert und es ist

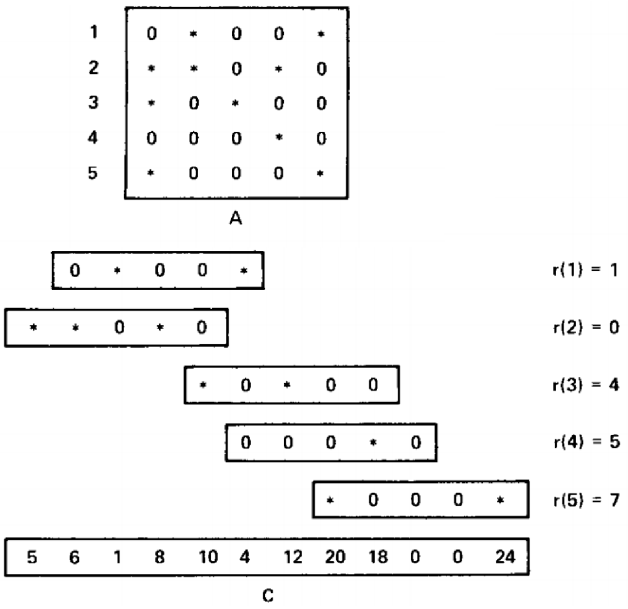


Abbildung 3.2.: Verschränkung von Zeilenvektoren nach [56] — Zellen mit einer Null markieren überflüssige Informationen. Das Ergebnis steht in C mit den entsprechenden Indizes der Originaltabelle A.

sichergestellt, dass sämtliche folgenden Wörter keine Änderungen an diesen Zuständen bewirken können. Da Zustände, die in die komprimierte Übergangstabelle eingefügt werden sollen, die Offsetpositionen[3] ihrer Nachfolgezustände kennen müssen, geschieht das Einfügen der Suffixzustände in umgekehrter Lesereihenfolge — also von links nach rechts. Der letzte, in eine solche Tabelle eingefügte Zustand ist immer der einzige Startzustand des Lexikon-Tries. Dieser Zustand repräsentiert das leere Wort ε, welches immer das längste gemeinsame Präfix aller Lexikoneinträge darstellt. Die Offsetposition dieses zuletzt eingefügten Zustands stellt dann den Eintrittspunkt zur späteren inkrementellen Suche in der Tabelle und im Trie dar.

Bei dem Verfahren werden die Zustände mitsamt all ihrer Übergänge und — im Falle der hier verwendeten Lexika — konzeptuellen Zuordnungen in der Tabelle als Zellen abgespeichert. Somit können Zellen entweder Zuständen im Trie entsprechen oder auch Übergängen im Trie. Die unterschiedlichen Zell-Typen verfügen dabei über unterschiedliche Zusatzinformationen, die zur Suche in der Tabelle benötigt werden.

[3] In einfachen Übergangstabellen ist dies nicht nötig, da in solchen die Zeilen- und Spaltengrößen fixiert sind und so die Offsetposition eines Zustandes direkt über die Alphabetgröße berechnet werden kann.

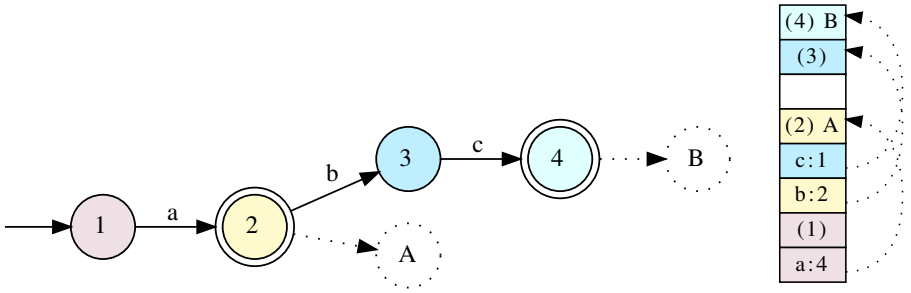


Abbildung 3.3.: Verschränkte Übergangstabelle eines Tries mit der Sprache $\mathcal{L} = \{a, abc\}$ mit der Typisierungsfunktion $t = \{(\langle a, A\rangle, \langle abc, B\rangle\}$. Die Pfeile in der Übergangstabelle deuten die Zielzustände der entsprechenden Übergangszellen an. Die geklammerten Zahlen in den Übergangszellen entsprechen den Zustandsnummern in der linken Abbildung.

Einfache, nicht finale Zustände benötigen keine weiteren Informationen. Finale Zustandszellen dagegen benötigen immer noch die konzeptuellen Zuordnungen. Zellen, die der Repräsentation von Übergangen dienen, benötigen sowohl die Offsetposition ihres Zielübergangs und auch den Buchstaben des Übergangs. Die Offsetposition des Ursprungsübergangs von Übergangszellen kann über die Offsetposition der Zelle und den numerischen Wert des zugeordneten Buchstaben mit ermittelt werden, indem die Offsetposition der Übergangszelle von dem numerischen Wert des Buchstabens abgezogen wird. Im Umkehrschluss dazu kann dieser Zusammenhang zwischen den Offsets der Übergangs- und Zustandszellen ausgenutzt werden, um die Offsetposition eines Übergangs von einem Zustand unter Konsum eines Buchstabens zu ermitteln. Dazu muss lediglich die Offsetposition des Zustandes mit der numerischen Zuordnung des Buchstabens summiert werden. Die Offsetposition des Zielzustands steht dann in der Übergangszelle an der resultierenden Offsetposition. Die Berechnung der Zielzustände kann somit, wie in der einfachen Übergangstabelle durch feste, in konstanter Zeit zu berechnende Offsetpositionen erfolgen. Eine ausführlichere Darstellung der Implementation der Zellen findet sich unter anderem in [19].

Beispiel 3.3.1. Ausgehend von dem, in Abbildung 3.3 dargestellten, Trie wird hier das Verfahren nochmal genau erläutert. Die beiden Lexikoneinträge *a* und *abc* werden in lexikographischer Reihenfolge in eine verschränkte Zustandsübergangstabelle eingefügt.

Das erste Wort in lexikographischer Reihenfolge ist *a*, das zweite ist *abc*. Erst wenn das zweite Wort eingefügt werden soll, kann das längste gemeinsame Präfix

und damit das längste einzutragende Suffix der beiden Worte berechnet werden. In diesem Beispiel ist das längste gemeinsame Präfix *a* und das längste einzutragende Suffix *bc*. Da in diesem Beispiel keine weiteren Einträge eingefügt werden und der erste Eintrag Präfix des zweiten ist, können nun die Übergänge von hinten nach vorne in die Übergangstabelle eingefügt werden.

Der erste Eintrag erfolgt nun für den finalen Zustand 4. Dieser Zustand hat keine ausgehenden Übergänge. Daher kann eine finale Zustandszelle in der Position 1 der Tabelle mit samt der konzeptuellen Zuordnung des Finalzustands (B) gespeichert werden. Als Ziel für den nächsten einzufügenden Zustand wird die Position des gerade eingefügten Zustands zwischengespeichert (1).

Als nächstes wird Zustand 3 in die Tabelle eingefügt. Der nächste freie Platz ist an Position 2 in der Tabelle. Dieser Zustand hat einen Übergang zu dem gerade eingefügten Zustand 4 an Position 1 mit dem Buchstaben *c*. Es wird somit eine Übergangszelle an der Position $2+c = 2+3 = 5$ eingefügt, die auf den Zielzustand an Position 1 in der Tabelle verweist.

Nun kann Zustand 2 in die Tabelle eingefügt werden. Dieser Zustand hat wiederum einen Übergang nach Zustand 3 mit dem Buchstaben *b*. Dieser Zustand kann nun nicht an der nächsten Position 3 eingefügt werden, da in diesem Fall die entsprechende Übergangszelle an der Position $3 + b = 3 + 2 = 5$ eingefügt werden müsste, die schon besetzt ist. Daher wird Zustand 2 an Position 4 und seine Übergangszelle an Position $4 + b = 4 + 2 = 6$ in der Übergangstabelle abgespeichert. Da Zustand 2 ebenfalls final ist wird dessen konzeptuelle Zuordnung (A) in der Zustandszelle gespeichert. Die Übergangstabelle verweist auf die Position des zuvor abgespeicherten Zustands (2).

Als Letztes wird der Startzustand des Tries mit seinem Übergang nach demselben Muster in die Tabelle an Position 7 eingefügt. Es ist anzumerken, dass auch Zustand 1 nicht an Position 3 gespeichert werden kann, da ebenfalls die Position der zugehörigen Übergangszelle mit einer bereits besetzten Zelle kollidiert. So entsteht eine leere Zelle an Position 3, die in diesem Fall nicht gefüllt werden kann.

Dabei gilt zu beachten, dass bei dieser Berechnung zusätzlich noch immer überprüft werden muss, ob an der berechneten Offsetposition tatsächlich eine Übergangszelle mit dem entsprechenden Buchstaben liegt. Ansonsten existiert kein Übergang und die Zelle an der berechneten Offsetposition repräsentiert entweder einen anderen Zustand des Tries, den Übergang eines anderen Zustands oder eine leere und somit unbesetzte Zelle.

Solche Leerzellen können in der berechneten Übergangstabelle auftreten. Sie markieren Positionen in der Tabelle, die mit keinem passenden Zeilenvektor besetzt werden konnten oder stellen Artefakte dar, die bei der Verschränkung zweier oder mehrerer Zeilenvektoren auftreten. Da der Algorithmus einen *first fit* und

keinen *best fit* Ansatz beim Einfügen der Zeilenvektoren verfolgt, kann das Auftreten vereinzelter, unbesetzter Zellen nicht ausgeschlossen werden. Bei der Verwendung von Tries zum besetzen der Tabelle können allerdings viele der auftauchenden Leerzellen durch bestimmte Finalzustände des Tries besetzt werden. Finalzustände, die keine weiteren Übergänge haben und somit den Blättern eines Baumes entsprechen, benötigen zur Speicherung nur eine einzige Zelle. Sie füllen somit automatisch viele der Leerpositionen auf. Nichtsdestotrotz stellt die erzeugte Übergangstabelle eine gute Approximation an eine ideale Verzahnung dar.

Abbildung 3.3 zeigt schematisch einen einfachen Trie (links) mit seiner zugehörigen verschränkten Übergangstabelle (rechts). Die Farben der Zellen markieren die Zustände und Übergänge der entsprechenden Zustände im Trie. Weiße Zellen stellen unbesetzte Leerzellen dar. Es ist gut zu erkennen, dass finale Zustände direkt ihre konzeptuellen Zuordnungen tragen und so eindeutig von nicht finalen Zuständen unterschieden werden können. Zellen, die Übergänge im Trie markieren sind immer mit dem entsprechenden Buchstaben des Übergangs und der Offsetposition des Zielzustands markiert. Man kann ihren zugehörigen Zustand finden, in dem man von deren Position aus genau so viele Zellen zurück geht, wie sie dem numerischen Wert des Buchstabens entsprechen. Der zentrale Einstiegspunkt in die Übergangstabelle stellt die Zelle des Startzustand des Tries an der Offsetposition 7 dar. Von diesem Punkt aus kann die Suche im Trie buchstabenweise ausgeführt werden.

Vergleicht man die Anzahl der Einträge der Übergangstabelle aus Abbildung 3.3 mit den $3 \times 4 = 12$ benötigten Zellen[4] der entsprechenden, einfachen Übergangstabelle, wird deutlich, dass schon bei diesem minimalen Beispiel eine Verkleinerung der benötigten Übergangstabelle erreicht werden kann, selbst wenn diese einzelne unbesetzte Stellen aufweist. Dieser Effekt würde sich bei entsprechend größeren und realistischeren Alphabeten sogar noch verstärken. Es gilt bei solchen Betrachtungen allerdings zu beachten, dass die Zellen der verschränkten Übergangstabelle im Allgemeinen etwas größer sind als die der einfachen Übergangstabelle. Somit ist die gewonnene Platzersparnis erst bei größeren Tries von echter Bedeutung.

Während Tries die gemeinsamen Präfixe aller ihrer Einträge unifizieren und somit jeweils nur einmal abspeichern, speichern sie gemeinsame Suffixe ihrer Einträge mehrfach ab. Im nächsten Abschnitt wird nun ein Verfahren dargestellt, welches in der Lage ist, die Anzahl der benötigten Zustände des Tries und somit die Größe der Lexika noch weiter zu reduzieren.

[4] die 4 Zustände des Tries multipliziert mit den 3 Buchstaben des Alphabets.

3.4. Von Tries zu minimierten Automaten

Um die Größe der als Trie repräsentierten Lexika noch weiter zu verkleinern, kann man versuchen, die Anzahl der benötigten Zustände noch weiter zu reduzieren. Dazu ist ein naheliegender Ansatz, die gemeinsamen Suffixe der Lexikoneinträge zu unifizieren und ebenso wie die gemeinsamen Präfixe der Lexikoneinträge nur einmal im Trie abzuspeichern.

Ein auf diese Weise bearbeiteter Trie wäre ein Automat, der sich von seinem Startzustand aus verästeln würde und sich dann wieder zu einem einzigen Finalzustand hin verjüngt. Ein solcher Automat hat im Allgemeinen immer weniger Finalzustände als Einträge. Das bedeutet auch, dass es nicht mehr so einfach möglich ist, die konzeptuellen Zuordnungen der Lexikoneinträge zu verwalten, da die eindeutige Zuordnung von Finalzuständen und Lexikoneinträgen verloren geht. Will man einen minimierten Automaten auch weiterhin zur konzeptuellen Zuordnung verwenden, wie es das hier behandelte Verfahren tut, muss bei einer Minimierung des Automaten darauf Rücksicht genommen werden, dass die konzeptuellen Zuordnungen der Einträge nicht verloren gehen.

Im weiteren Verlauf der Betrachtungen wird nun zuerst auf ein Verfahren eingegangen, welches aus einem Trie einen wie eben beschriebenen minimierten Automaten erzeugt und somit keine weiteren konzeptuellen Zuordnungen mehr zulässt. Danach wird auf die nötigen Erweiterungen eingegangen, die es ermöglichen den Trie in einen teilweise minimierten Automaten umzuformen, indem gemeinsame Suffixe genau so unifiziert werden, dass dabei die konzeptuellen Zuordnungen nicht verloren gehen.

3.4.1. Minimierte Automaten

In [11] wird ein Verfahren erläutert, das einerseits dazu dient, einen Trie zu minimieren und andererseits dazu in der Lage ist, eine verschränkte Übergangstabelle, wie in Kapitel 3.3 dargestellt, sequenziell aufzubauen. Das Verfahren läuft dabei in lexikalischer Reihenfolge über die Lexikoneinträge und baut dabei sequenziell eine verschränkte Übergangstabelle auf dieselbe Weise auf, wie es das bereits erläuterte Verfahren tut.

Die grundlegende Erweiterung, die zur Minimierung des Automaten führt, ist dabei ein spezielles Register, welches alle bereits eingetragenen Zustände enthält und bei jedem Einfügen neuer Zustände in die Tabelle entscheidet, ob ein neuer Eintrag eingefügt werden muss oder ob sich schon ein äquivalenter Zustand in der Tabelle befindet. Im ersten Fall wird der neue Zustand einfach in die Tabelle eingetragen. Dies kann dann auf dieselbe Weise geschehen, auf die Zustände schon im normalen Verfahren in die Tabelle eingefügt werden. Die Offsetposition des

Zustandes und der Zustand selbst werden dann zum späteren Nachschlagen im Register abgelegt. Befindet sich bereits ein äquivalenter Zustand im Register, wird der neue Zustand nicht mehr neu eingefügt und es wird lediglich die im Register abgelegte Offsetposition — die als Zielposition für weiter links kommende Zustände verwendet wird — des äquivalenten Zustandes in der Tabelle verwendet. Das Register speichert somit äquivalente Zustände und verhindert so, dass Zustände unnötigerweise mehrfach in die Tabelle eingefügt werden.

Um ein schnelles Nachschlagen äquivalenter Zustände zu gewährleisten, kann das Register als Hash implementiert werden. Dies ermöglicht eine amortisiert konstante Komplexität zum Nachschlagen von Zuständen und verbessert die Laufzeit des Algorithmus.

Das Register verwendet zur Überprüfung der Äquivalenz von Zuständen dabei Äquivalenzklassen über die Zustände des Automaten, wie sie in Lemma 1.3.2 dargestellt ist. Diese führt dann im Algorithmus dazu, dass alle Zustände, von denen aus nur gleiche Suffixe akzeptiert werden, eindeutig im Register abgelegt und erkannt werden können. Somit werden nur Zustände aus neuen, noch nicht vorgefundenen Zuständen in die Tabelle eingefügt. Dieses Vorgehen entspricht der vorwärts gerichteten Einfügung von Zuständen in den Automaten unter der Beachtung der Suffixe der Wörter. Durch die einmalige Einfügung der Äquivalenzklassen werden die gemeinsamen Suffixe unifiziert und müssen nur noch ein einziges mal im Automaten gespeichert werden. Diese Unifizierung führt zu einer Reduktion der benötigten Zustände und somit direkt zu einer Verkleinerung der zur Darstellung des Tries benötigten Übergangstabelle.

Abbildung 3.4 stellt abschließend einen minimierten Automaten dar. Der minimierte Automaten unifiziert hier nicht mehr nur die gemeinsamen Präfixe der Einträge, sondern auch die gemeinsamen Suffixe. Da der minimierte Automat in diesem Beispiel nur über einen Finalzustand verfügt, ist klar ersichtlich, dass eine konzeptuelle Zuordnung über die finalen Zustände des Automaten, wie sie in Abbildung 3.1 des normalen Lexikon-Tries dargestellt wird, nicht mehr möglich ist. Somit eignen sich minimierte Automaten nicht mehr zur Speicherung von konzeptuellen Zuordnungen und eignen sich nicht zu der hier benötigten Verwendung von Lexika.

3.4.2. Minimierung mit Erhalt konzeptueller Zuordnungen

Die eben vorgestellte Minimierung von Tries hat für das hier benötigte Verfahren den Nachteil, dass bei einer solchen Minimierung die Möglichkeit der konzeptuellen Zuordnung der Einträge verloren gehen würde. Der Minimierungsalgorithmus kann allerdings auf einfache Weise erweitert werden, um einerseits eine Verringerung der Zustandsmenge zu erreichen und andererseits eine weitergehende Zuord-

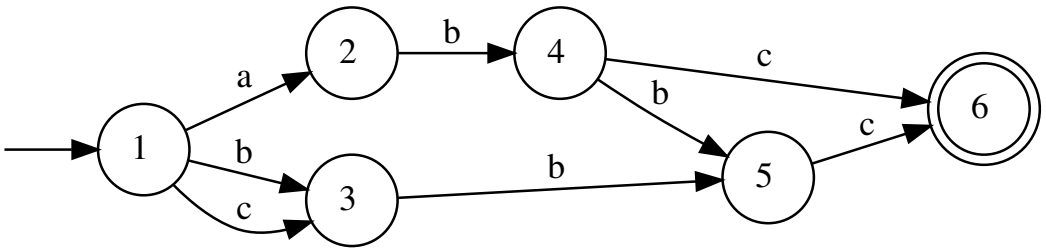


Abbildung 3.4.: Minimierter Trie der Sprache $\mathcal{L} = \{abc, abbc, bbc, cbc\}$

nung zu gewährleisten. Es ist offensichtlich, dass dieser erweiterte Algorithmus die Zustandsmenge nur in einem geringeren Maß verringern kann.

Will man einen minimierten Automaten erzeugen, ohne die an den Finalzuständen gespeicherte Information zu verlieren, können offensichtlicherweise nur diejenigen gemeinsamen Suffixe von Einträgen unifiziert werden, die über dieselben Daten und somit über die gleichen konzeptuellen Zuordnungen verfügen. Solch eine Form der Minimierung kann allerdings nur dann zu einer Reduktion der Größe führen, wenn die Anzahl der Lexikoneinträge deutlich größer ist als die Anzahl der von diesen referenzierten Konzepte. Entspricht die Anzahl der Lexikoneinträge annähernd der der Konzepte, wird eine solche Minimierung zu kaum einer Reduktion der Größe führen. In solch einem Szenario würden sich die konzeptuellen Zuordnungen fast aller Lexikoneinträge unterscheiden und somit nur eine geringe Anzahl von Lexikoneinträgen unifiziert werden.

Abbildung 3.5 zeigt einen einfachen Automaten, dessen Suffixe mit gleichen Zuordnungen unifiziert wurden. Vergleicht man diesen Automaten mit den Abbildungen des einfachen Tries aus Abbildung 3.1 auf Seite 50 und mit der Abbildung des minimierten Automaten in Abbildung 3.4 auf Seite 59 wird deutlich, dass die Anzahl der Zustände geringer ist, als die des einfachen nicht minimierten Tries, die Reduktion der Zustandsmenge des minimierten Automaten dagegen nicht erreicht werden kann. Der dargestellte Automat unifiziert, im Gegensatz zum minimierten Automaten, die gemeinsamen Präfixe und Suffixe der Einträge genau dann, wenn sie eine identische, konzeptuelle Zuordnung in ihrem Finalzuständen besitzen.

Der Algorithmus, der zum Aufbau eines solch minimierten Automaten verwendet wird, entspricht weitestgehend dem aus dem vorhergegangenen Abschnitt. Es wird einzig eine Änderung in der in Lemma 1.3.2 dargestellten Definition der Äquivalenzklassen von Zuständen benötigt.

Definition 3.4.1 (Äquivalenz von Zuständen 2). Sei $\langle \Sigma, Q, s, \delta, F, \theta, t \rangle$ ein $\mathcal{W}$-Automat im Sinne von Definition 1.5.3. Zwei Zustände $p, q \in Q$ sind äquivalent genau dann,

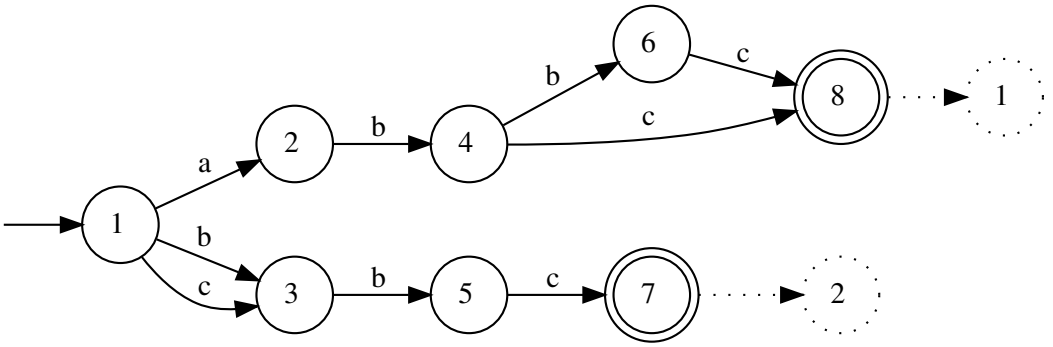


Abbildung 3.5.: Teilweise minimierter Automat der Sprache $\mathcal{L} = \{abc, abbc, bbc, cbc\}$ mit den konzeptuellen Zuordnungen $\iota = \{(abc, 1), (abbc, 1), (bbc, 2), (cbc, 2)\}$.

wenn sie äquivalent im Sinne der Definition 1.3.2 sind und wenn gilt:

$$p \in F \wedge q \in F \implies \theta(p) = \theta(q)$$

Die Anwendung des Minimierungsalgorithmus kann dann auf dieselbe Weise wie vorher ablaufen, indem die einzufügenden Zustände in einem Register abgelegt werden und alle äquivalenten Zustände nur einmal in die Übergangstabelle eingefügt werden. Da nun finale Zustände nur dann äquivalent sind, wenn sie auch über dieselbe konzeptuelle Zuordnung verfügen, können nun Finalzustände nur dann unifiziert werden, wenn sie über dieselben konzeptuellen Zuordnungen verfügen.

Wie eingangs bereits erwähnt, hängt die Effizienz dieser Form der Minimierung sehr stark von der Struktur der Sprachressourcen und den internen Verbindungen der Daten untereinander ab. Gerade wenn die Anzahl der Konzept und die der Lexikoneinträge nah beieinander liegen, kann dieses Verfahren nur sehr schlecht die Anzahl der Zustände und Übergänge im Lexikon verringern. Solche Eigenschaften sind in realistischen Anwendungsszenarien allerdings sehr unwahrscheinlich, da die Anzahl der natürlichsprachlichen Ausdrücke, die ein abstraktes Konzept referenzieren, deutlich größer ist als die Anzahl der abstrakten Konzepte selbst. Dies ist insbesondere auch dann der Fall, wenn diese Automaten zur Speicherung von Vollformenlexika verwendet werden und viele unterschiedlich flektierte Formen einzelner Lexikoneinträge in den Lexika abgelegt werden.

In realistischeren Anwendungsszenarien ist eher davon auszugehen, dass die Anzahl der Konzepte deutlich geringer ausfällt als die Anzahl der Lexikoneinträge. Einzelne Konzepte können meistens durch mehrere unterschiedliche, natürlichsprachliche Muster identifiziert werden. Somit übersteigt die Anzahl der Lexi-

koneinträge die der Konzepte. In solchen Fällen kann der Minimierungsalgorithmus im Allgemeinen zumindest einige der gemeinsamen Suffixe und zugehörigen Finalzustände[5] vereinigen.

Die Lexika ermöglichen explizit die Behandlung von Mehrwortverbindungen. Bei den Betrachtungen der Minimierungsmöglichkeiten von Suffixen sollte nicht vergessen werden, dass es sich bei den Suffixen der Einträge nicht nur um einzelne zusammenfallende morphologische Muster handelt, sondern insbesondere auch um überlappende Wörter komplexer Mehrwortverbindungen. Bei solchen Lexikoneinträgen besteht ein erhebliches Potential für den Algorithmus, die Anzahl der benötigten Zustände und damit direkt die Größe der Lexika zu verkleinern, da die Vereinigung solcher längeren Verbindungen allein schon eine Menge überflüssiger Zellen einsparen kann.

Abbildung 3.6 zeigt die vereinfachte Darstellung eines realistischen, minimierten Lexikons. Bei der Abbildung wurde darauf verzichtet, unnötige Zustände und Zustandsübergänge darzustellen. Diese werden nur über die Label der zu erkennenden Teilmuster markiert[6]. Bei Mustern von Lexikoneinträgen, wie sie diesem Beispiel dargestellt sind, bestehen einige Möglichkeiten für den Algorithmus, bestimmte Suffixe wirklich nur einmal abspeichern zu müssen. Gerade Eigennamen von Personen und Organisationen, mit optionalen Erweiterungen wie Berufsbezeichnungen und Aufgabenbereiche, referenzieren oft eindeutige gemeinsame Konzepte. Hier kann eine nicht zu unterschätzende Verringerung der Zustandsmenge erreicht werden, da das wiederholte Abspeichern der Zustandsfolge zur Erkennung eines mehrfach auftauchenden Namens entfallen kann.

3.5. Vergleich der Minimierungsalgorithmen

In diesem Abschnitt werden nun die gerade diskutierten Techniken zur Speicherung von Lexika analysiert. Dabei wird die Größe der Zustandsmenge der Automaten untersucht, die benötigt werden, um ein realistisches Lexikon mitsamt den konzeptuellen Zuordnungen abzuspeichern.

Für die Untersuchungen wurden zwei unterschiedliche Ontologien verwendet. Zum einen wurde ein Ausschnitt des TopicZoom EFGT-Netzes (vgl. Kapitel 4.3) verwendet. Dieser für die Untersuchungen verwendete Ausschnitt umfasst 10.334 Konzepte, die von 24.770 eindeutigen Lexikoneinträgen referenziert werden.

[5] Diese einfache Form der Unifizierung kann der Algorithmus in den meisten Fällen, unabhängig von der Anordnung der Daten und deren Suffixüberschneidungen vornehmen.

[6] Hierbei gilt zu beachten, dass bei den tatsächlichen Automaten diese Label sehr wohl als einzelne Zustandsübergänge mit den entsprechenden Zeichen des Alphabets repräsentiert sind.

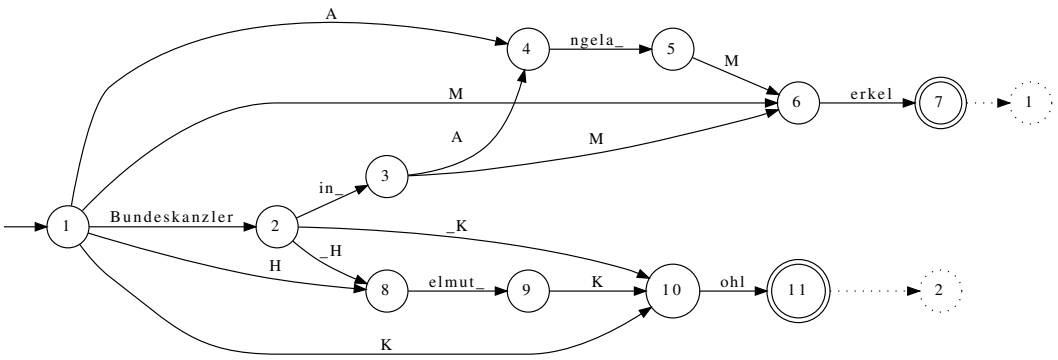


Abbildung 3.6.: Vereinfachte Darstellung eines Lexikons als minimierter Automat aus den Einträgen *Angela␣Merkel*, *Bundeskanzlerin␣Angela␣Merkel*, *Bundeskanzlerin␣Merkel*, *Merkel*, *Helmut␣Kohl*, *Bundeskanzler␣Helmut␣Kohl*, *Bundeskanzler␣Kohl* und *Kohl* mitsamt den entsprechenden, natürlichen konzeptuellen Zuordnungen.

Zum anderen wurde auch die, für die Domäne des ersten Weltkriegs spezialisierte Ontologie „Erster Weltkrieg“ (vgl. Kapitel 4.5.2) ausgewertet. Diese Ontologie umfasst Personen, Ereignisse und Geoentitäten, die in Texten über den 1. Weltkrieg von Bedeutung sind. Sie umfasst 3626 verschiedene Konzepte und 10.670 eindeutige, zum Teil sehr lange Lexikoneinträge in den drei Sprachen Deutsch, Englisch und Russisch.

Aus Lexika der beiden Wissensressourcen wurden jeweils ein einfacher Trie, ein teilweise minimierter Automat und ein vollständig minimierter Automat erzeugt und die Anzahl der Zellen in der verschränkten Übergangstabelle der resultierenden Automaten gezählt. Bei der Zählung der Zellen wurden alle Zellen des Automaten, inklusive der Übergangszellen und etwaiger Leerzellen, berücksichtigt. Der vollständig minimierte Automat dient in dieser Darstellung als natürliche untere Schranke für die mit diesem Verfahren zu erreichende Minimierung, die aber für die hier angestrebte Verwendung zur Abbildung von Lexikoneinträgen auf Konzepte nicht zu gebrauchen ist. Auf eine Angabe des benötigten Speicherplatzes der Übergangstabellen wurde verzichtet, da diese stark von den verwendeten Speicherungstechniken, der implementationsabhängigen Größe der Zellen und möglicher weiterer Komprimierungsverfahren abhängen. Die Anzahl der Zellen reicht als Indikator für die tatsächlich benötigte Größe der Übergangstabellen im Speicher vollkommen aus.

Tabelle 3.2 listet die Zahlen der Zellen mitsamt der prozentualen Verkleinerung in der Anzahl der Zellen auf. Die Größe der Übergangstabelle der Automaten, die

Wissensressource	einfach	teilweise-minimiert	minimiert
TopicZoom EFGT-Netz	481.295	375.628	178.059
Ontologie „Erster Weltkrieg”	381.385	266.617	198.670
Faktor TopicZoom EFGT-Netz	1,0	0,78	0,40
Faktor Ontologie„Erster Weltkrieg”	1,0	0,70	0,52

Tabelle 3.2.: Überblick über die Anzahl der Zellen von Übergangstabellen verschiedener Lexikonautomaten

mit Erhaltung der konzeptuellen Zuordnungen minimiert wurden, liegt dabei für beide analysierten Wissensressourcen ungefähr in der Mitte zwischen dem einfachen Trie und dem vollständig minimierten Automaten. Diese hier erreichte, deutliche Größenreduktion gegenüber des einfacheren Tries rechtfertigt in den meisten Fällen die Anwendung des dargestellten Minimierungsalgorithmus, auch wenn die Erzeugung der minimierten Lexika aufgrund der zusätzlich benötigten Hilfsstrukturen aufwendiger in Hinblick auf Hauptspeicherverbrauch und Laufzeit ist.

In Hinblick auf diese Nachteile bei der Erzeugung minimierter Lexikon-Tries sei zusätzlich noch darauf hingewiesen, dass die Lexika für das Verfahren nur einmal beim Aufbau der Wissensressource und der Lexika aus den Eingabedateien erzeugt werden. Bei der späteren Dokumentenanalyse müssen die Wissensressourcen nicht mehr jedes Mal neu erzeugt werden. Vielmehr profitieren diese späteren Schritte von der Größenreduktion der verwendeten Lexikonautomaten.

3.6. Zusammenfassung

In diesem Kapitel wurden die unterschiedlichen computerlinguistischen Techniken, die zur internen Repräsentation von Lexika von Wissensressourcen zur semantischen Indexierung nötig sind, dargestellt. Die Lexika bilden die Brücke zwischen natürlichsprachlichen Dokumenten und den abstrakten Konzepten der Wissensressource. Sie bilden einen zentralen Punkt bei der semantischen Indexierung, da sie zur Identifizierung der Konzepte in den Dokumenten verwendet werden.

Um eine effiziente Suche auf Dokumenten einerseits und eine Handhabung großer Vollformenlexika andererseits gewährleisten zu können, werden die Lexika der Wissensressource der semantischen Indexierung als speziell minimierte Automaten repräsentiert, die alle gemeinsamen Präfixe der Einträge unifizieren und ebenso die gemeinsamen Suffixe zusammenfassen, ohne dabei die konzeptuellen Zuordnungen zu verlieren. Die Lexikoneinträge sind in der Anwendung direkt mit den Knoten des Graphen der Wissensressource verbunden, um eine schnelle Ver-

bindung zwischen den natürlichsprachlichen Lexikoneinträgen und den abstrakten Konzepten der Wissensressource zu ermöglichen.

4. Explizite Wissensressourcen

Explizite Wissensressourcen dienen dazu, menschliches (Welt-) Wissen zu formalisieren und für Maschinen nutzbar zu machen. Die semantische Indexierung verwendet explizite Wissensressourcen, um den Index der Dokumentenkollektion mit dem in den Wissensressourcen abgebildete Wissen anzureichern. Dieses Kapitel beschreibt verschiedene Formalismen zur Repräsentation von Wissen und gibt ebenso zwei konkrete Beispiele für real existierende Wissensressourcen, die in den nachfolgenden Kapiteln als Beispiele für alle weiteren Betrachtungen zur semantischen Indexierung dienen werden.

4.1. Wissensressourcen

Im Allgemeinen wird Wissen durch die Verbindungen zwischen unterschiedlichen Konzepten dargestellt. Die Menge der Konzepte und deren Verbindungen bilden dabei verschiedene Relationen im mathematischen Sinne. Sie besteht aus einer binären homogenen Relation auf der Menge der abstrakten Konzepte der Wissensressource. Zusätzlich zu den direkt vorhandenen, explizit ausgezeichneten Verbindungen kann man weitere *abgeleitete Relationen* betrachten, die sich axiomatisch, zum Beispiel mit Hilfe von Transitivitäts- oder Symmetriebetrachtungen, definieren lassen. Es ist offenkundig, dass die Konzepte der Wissensressourcen — jenseits von natürlichsprachlichen Mehrdeutigkeiten — immer eindeutig sein müssen (vgl. Definitionen 1.5.1 und 1.5.3). Diese Forderung wird von verschiedenen Formalismen zur Darstellung von Wissensressourcen auf unterschiedliche Weise umgesetzt.

Die in dieser Arbeit behandelte semantische Indexierung mit expliziten Wissensressourcen verwendet Wissen, welches durch solche verschiedene Wissensressourcen repräsentiert wird, zur Indexierung von natürlichsprachlichen Texten und zur Anreicherung des Index mit dem explizit in den Wissensressourcen abgespeicherten Verbindungen. Dazu müssen die in den Wissensressourcen dargestellten Fakten analysiert und dann in eine zur Indexierung maschinell verarbeitbare Form überführt werden.

Die verschiedenen Wissensressourcen stellen das kodierte Wissen auf unter-

schiedliche Weise dar und folgen bei der Erstellung der Ressourcen unterschiedlichen Formalismen. Damit die Indexierung mit unterschiedlichen Wissensbasen arbeiten kann, müssen unterschiedliche Formalismen auf mögliche Gemeinsamkeiten untersucht werden. Diese Gemeinsamkeiten in der Darstellung der Ressourcen führen zu einer generalisierten Darstellung von Wissen, welche schließlich in die nötigen Datenstrukturen für die semantische Indexierung überführt werden kann.

Um auch abgeleitete Relationen effizient beim Indexieren berücksichtigen zu können, müssen verschiedene implizite Verbindungen in den Wissensressourcen vor der Indexierung explizit ausgezeichnet worden sein. Hierzu werden verschiedene Hüllenbildungen (vgl. Kapitel 1.1.2) auf den Relationen der Wissensressource ausgeführt, die es dann ermöglichen, implizite Verbindungen explizit in den Datenstrukturen zur semantischen Textindexierung auszuzeichnen und zu verwenden.

In den folgenden Abschnitten werden eine Reihe von expliziten Wissensressourcen behandelt, die als Wissensbasis zur semantischen Textindexierung herangezogen werden können. Dabei werden verschiedene Gemeinsamkeiten und Unterschiede der Ressourcen dargestellt und abschließend eine hinreichende Abstraktion vorgestellt, die es ermöglicht, die verschiedenen Arten von Wissensressourcen uniform behandeln zu können. Es werden in diesem Kapitel bei weitem nicht alle Arten von Wissensressourcen behandelt. Einen genaueren Überblick über die verschiedenen Arten von Wissensressourcen kann unter anderem [10] geben.

4.2. Konzepte und Uniform Resource Identifier

Wie in der Einleitung der Arbeit bereits erwähnt wurde, sind die Konzepte in der Wissensbasis eindeutig. Jedes Konzept kann somit immer eindeutig über eine diesem Konzept zugeordneten *Uniform Resource Identifier (URI)* [25] identifiziert werden. Diese Bezeichner werden immer mehr zur einheitlichen und eindeutigen Identifizierung von abstrakten und physikalischen Ressourcen eingesetzt. Sie dienen vorrangig der Bezeichnung von E-Mail-Adressen, Webseiten und anderen Daten im Internet (Links), können aber auch nur zur einheitlichen Unterscheidung von Konzepten verwendet werden. Es ist nicht zwingend erforderlich, dass URIs von Konzepten auf tatsächlich vorhandene Ressourcen im Internet verweisen.

Verschiedene Wissensressourcen verwenden unterschiedliche Arten von URIs für die Konzepte. Einige verwenden Links zu entsprechenden Einträgen in der Wikipedia oder zu Normdaten wie z.B. VIAF (The Virtual International Authority File)[1]. Andere verwenden wiederum eigene Schemata zur Generierung eindeutiger Identifizierer.

[1] `https://viaf.org/`

Bei den Betrachtungen von Wissensressourcen spielt die genaue Art oder die Bedeutung der URI keine Rolle. Ebenso spielt es kein Rolle, ob die verwendeten URIs auch tatsächlich syntaktisch valide URIs darstellen oder ob sie tatsächlich irgendwelche Ressourcen referenzieren. Lediglich die Eindeutigkeit der verwendeten Identifikatoren ist wichtig. Für die Anwendung der Wissensressource zur semantischen Indexierung reicht *lokale* Eindeutigkeit innerhalb der verwendeten Wissensressource aus[2]. Dennoch verwenden die im weiteren Verlauf noch dargestellten Wissensressourcen valide, global eindeutige URIs zur Identifikation eindeutiger Konzepte.

Nur wenn bestehende Systematiken zur Auszeichnung von Wissensressourcen verwendet werden sollen, oder wenn mehrere Wissensressourcen zusammengefügt werden sollen, müssen die verwendeten URIs auch *global* eindeutig sein. Dies kann zumeist auch nachträglich durch die Konkatenation eines global eindeutigen Namensraumpräfix [43, S.28] mit den lokal eindeutigen URIs geschehen. Weitere Informationen zu URIs und den verwandten Konzepten URLs und URNs kann man unter anderem in [43, S.26ff] finden.

Für Wissensressourcen, die selbst über kein Konzept von URIs verfügen, können auf einfache Weise künstliche URIs erstellt werden, indem deren interne Identfizierungsmechanismen über eindeutige Web-Domänen in *echte* URIs überführt werden. Nur in Ausnahmefällen müssen auch vollkommen neue URIs für die Konzepte der verwendeten Wissensressource erstellt werden.

4.3. EFGT-Netze

EFGT-Netze [10, 9, 50] stellen einen Formalismus zur Abbildung geographischer, zeitlicher und thematischer Aspekte dar. Ihr Name leitet sich von ***Entities,*** thematic ***Fields, Geographic and Temporal areas*** ab[3]. Der Formalismus wurde gezielt zur Repräsentation von semantischem Wissen entwickelt und unterstützt explizit die maschinelle Weiterverarbeitung zur automatischen Indexierung, die Suche und Klassifikation von natürlichsprachlichen Texten.

Die Einträge des Netzes stammen aus einer von drei Haupthierarchien, *thematische Felder*, *geographische Zonen* und *zeitliche Zonen*. Diese Einträge und deren Verbindungen untereinander bilden gemeinsam einen *azyklischen Graphen* mit einem absoluten *Wurzelknoten* bzw. *Topnode*. Dieser Wurzelknoten stellt das allgemeinste Konzept der Hierarchie dar. Da in dem Netz Verbindungen zwischen Elementen unterschiedlicher Haupthierarchien möglich sind, bilden alle Einträge zusammen

[2] Jedenfalls solange bis die verwendeten Wissensressourcen nicht durch externe Wissensressourcen erweitert werden sollen.

[3] Deutsch: Objekte, thematische Felder, geographische und zeitliche Zonen

keine strenge Hierarchie von Konzepten, sondern ein komplexes Geflecht aus zeitlichen, thematischen und geographischen Gebieten.

Jede der drei Hauptachsen (vgl. oben) bildet eine Taxonomie von semantisch breiteren Themengebieten hin zu semantisch engeren. *Benannte Entitäten* können in diesen thematisch, geographisch und temporalen Raum eingeordnet werden [10, S.42]. Auf diese Weise kann der geographische, thematische und zeitliche Kontext von Einträgen klar definiert werden.

Beispiel 4.3.1. Die beiden Persönlichkeiten *Angela Merkel* und *Barak Obama* werden beide thematisch unter anderem unterhalb der Felder *Politik* und *Personen und Persönlichkeiten* sowie unterhalb eines aktuellen zeitlichen Kontextes eingeordnet, wohingegen sich ihre jeweiligen geographischen Einteilungen (*Deutschland* vs. *Vereinigte Staaten von Amerika*) unterscheiden.

Alle Einträge eines EFGT-Netzes gehören zu einem von vier Typen, die hier im folgenden aufgelistet werden [10, S. 46].

thematisch Thematische Einträge umfassen abstrakte Konzepte, von sehr allgemeiner Natur, wie z.B. *Kunst* oder *Politik*, bis hin zu speziellen Themen wie *Internetprogrammierung* oder *Surrealismus*.

geographisch Geographische Einträge beinhalten allgemeine geographische Gebiete, wie z.B. *Bayern*. Aber auch geographische Sammelbegriffe wie etwa *Berge* oder *Flüsse*.

temporal Temporale Einträge umfassen Zeitpunkte und Zeiträume unterschiedlichster Arten. Sie umfassen Erdzeitalter wie *Paläozotikum* oder *Mesozoikum*, Jahrhunderte, Jahrzehnte und Jahre sowie Perioden wie das *Mittelalter* oder die *Renaissance*.

entiätsspezifisch Entitätsspezifische Einträge beschreiben Mengen von Entiäten. Einträge die Personen, Organisationen und Ereignisse darstellen, gehören zu diesem Typ.

Beispiel 4.3.2. Der komplexe Ausdruck *deutsche Literatur des 19. Jahrhunderts* könnte so dargestellt werden, indem das Konzept mit dem Konzept *Literatur* aus der Hierarchie der thematischen Felder, dem Konzept *Deutschland* aus der geographischen Hierarchie und mit dem Konzept *19. Jahrhundert* aus der temporalen Hierarchie verbunden wird (vgl. [9, S. 2]).

Über die Verbindungen mit den drei unterschiedlichen Hierarchien entsteht eine verwobene Netzstruktur. Abbildung 4.1 zeigt wie dieses Konzept in die drei

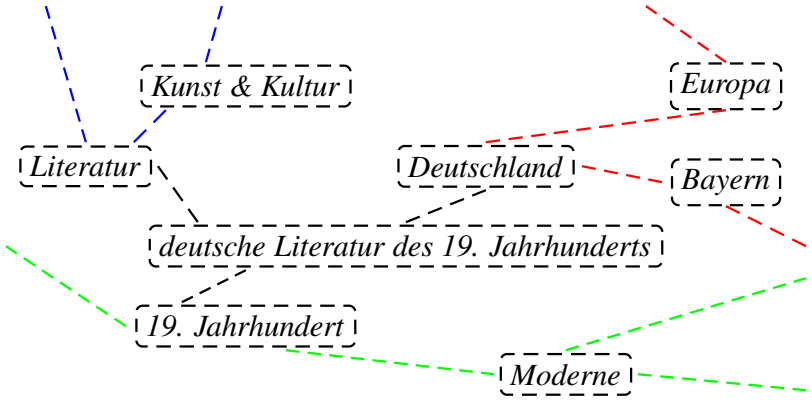


Abbildung 4.1.: Ausschnitt eines hypothetischen EFGT-Netzes für das Konzept *deutsche Literatur des 19. Jahrhunderts*. Blaue Verbindungen deuten thematische Verbindungen, grüne temporale Verbindungen und rote geographische Verbindungen an. Schwarze Verbindungen zeigen die entitätsspezifischen Einordnungen an.

Hauptachsen einsortiert werden kann und visualisiert das resultierende Netz. Die farbigen Verbindungen deuten die ursprünglichen Hauptachsen der drei Haupthierarchien an.

Aus der Abbildung geht auch hervor, dass durch die Verbindung der drei transitiven Hierarchien die der Semantik des Begriffs zugrundeliegende Logik gut nach gezeichnet wird. Somit kann der Begriff *deutsche Literatur des 19. Jahrhunderts* durchaus ebenso als eine Instanz von *deutsche Literatur der Moderne* oder sogar auch als Instanz von *europäische Kunst & Kultur der Moderne* angesehen werden.

In Kombination bilden diese drei Hauptachsen aber nicht nur einen einfachen orthogonalen dreidimensionalen Raum ab, in denen Einträge im Hinblick auf einzelne thematische, geographische und zeitliche Aspekte einsortiert werden. Es ist vielmehr so, dass die Achsen auf unterschiedliche Weise kombiniert werden können, um komplexe semantische Beziehungen abzubilden.

4.3.1. Hüllenbildung auf EFGT-Netzen

EFGT-Netze spannen gerichtete azyklische Graphen auf. Die direkten Verbindungen zwischen einzelnen Einträgen des Netzes sind dabei in einer Relation abgespeichert. Es sind somit nur die direkten Verbindungen der Einträge untereinander explizit abgespeichert, nicht aber die impliziten transitiven Ableitungen.

Da die Hierarchie, die EFGT-Netze abbilden, für *„ist allgemeiner als"* steht und

damit implizit transitiv ist (vgl. dazu auch Beispiel 4.3.2), aber eben nur die direkten Verbindungen von Konzepten untereinander explizit ausgezeichnet sind, sollen zur semantischen Indexierung diese transitiven Ableitungen explizit mit in den Indexierungsprozess aufgenommen werden. Diese weiteren transitiven Verbindungen sind ja eben ein wichtiger Teil des in der Ressource abgespeicherten Wissens.

Um diese impliziten Verbindungen explizit in der Relation auszuschreiben, muss lediglich die den Netzen zugrunde liegende Relation transitiv abgeschlossen werden. Durch die transitive Hüllenbildung der Relation sind dann alle Verbindungen explizit in der Relation ausgeschrieben. So kann bei der Indexierung mit semantischen Wissensressourcen direkt auf alle Verbindungen einzelner Konzepte zugegriffen werden.

Bei der Betrachtung von EFGT-Netzen wird immer implizit die transitive Hülle der ausgezeichneten Relation betrachtet. Da der durch die EFGT-Netze aufgespannte Graph azyklisch ist und Unterthemen nur direkt mit ihren Oberthemen verbunden sind, folgt, dass die Relation von EFGT-Netzen auch antisymmetrisch ist.

4.3.2. TopicZoom

Die erste im Rahmen dieser Arbeit zu Testzwecken verwendete Wissensressource ist das EFGT-Netz der Firma *TopicZoom*. Das *TopicZoom Netz* wurde von dem 2008 als Spin-off des Centrums für Informations- und Sprachverarbeitung (CIS) der Ludwig-Maximilians-Universität München (LMU) gegründetem Unternehmen TopicZoom[4] entwickelt und aufgebaut.

Dieses Netz bildet die Grundlage zur semantischen Indexierung, wie sie schon in [19] beschrieben ist. Ausgehend von einem in diesem Kapitel vorgestellten Beispiel wird später noch das genaue Vorgehen bei der semantischen Indexierung dargestellt.

Die Gründer und Mitarbeiter der Firma waren allesamt an der Entwicklung, Ausarbeitung und Erstellung von EFGT-Netzen beteiligt. Die Firma entwickelt unter anderem eine Reihe von Werkzeugen, welche die einfache Erstellung, Erweiterung und Abfrage von EFGT-Netzen ermöglicht. Ebenso beschäftigt sie sich mit der aktiven Weiterentwicklung und manuellen Erweiterung des TopicZoom Netzes.

Das TopicZoom Netz enthält Wissen zu aktuellen und historischen Wissen- und Themengebieten. Es enthält Wissen über Persönlichkeiten des öffentlichen Lebens aus verschiedenen Lebensbereichen, darunter unter anderem die Bereiche *Sport*,

[4] `http://www.topiczoom.de`

Politik oder *Kunst und Kultur*, aber auch Informationen zu öffentlichen Einrichtungen, zeitlichen Gliederungen und eine politische und geologische Einteilung der Welt.

Hier wird nun anhand eines kleinen Beispiels die allgemeine Struktur des TopicZoom Netzes und somit auch die Struktur von EFGT-Netzen verdeutlicht. Dieser Auszug aus dem TopicZoom Netz wird im weiteren Verlauf der Arbeit für verschiedene Beispiele herangezogen, um verschiedene Aspekte der semantischen Indexierung mit expliziten Wissensressourcen genauer darstellen zu können.

Das TopicZoom Netz ist in dem auf XML basierenden Formalismus RDF [1] ausgezeichnet, der in Kapitel 4.6.1 noch genauer behandelt werden wird. Jedes Konzept im Netz ist als eigener Knoten unterhalb der Wurzel repräsentiert. Alle Konzepte im Netz sind über URIs eindeutig identifiziert. Zusätzlich zu ihrer URI haben sie zumeist einen ausgezeichneten natürlichsprachlichen Vorzugsnamen, welcher der einfacheren Identifikation der jeweiligen Konzepte dient und optional weitere alternative Namen, die verschiedenen Flexionsformen, Schreibvarianten und Synonyme des jeweiligen Konzeptes enthalten. Diese Namen sollen die Identifikation der abstrakten Konzepte des Netzes in natürlichsprachlichen Texten direkt unterstützen und vermeiden so gezielt Ambiguitäten.

Die Verbindungen zu anderen Konzepten der TopicZoom Hierarchie, mit denen ein Konzept jeweils direkt verbunden ist, sind für jeden einzelnen Begriff angegeben. Zur eindeutigen Identifizierung der Konzepte werden immer eindeutige Konzept-URIs des Netzes und nicht deren Vorzugsnamen verwendet. Dies erschwert zwar einerseits das direkte Bearbeiten der Hierarchie durch menschliche Benutzer, erleichtert aber deren maschinelle Handhabung enorm. Durch die Verwendung von natürlichsprachlichen Vorzugsnamen kann die Hierarchie aber auch von menschlichen Benutzern unter der Zuhilfenahme von verschiedenen spezialisierten Werkzeugen bearbeitet werden [10].

Abbildung 4.2 zeigt einen Ausschnitt der TopicZoom Hierarchie zu den beiden Einträgen mit dem Vorzugsnamen *Michael Bloomberg* und *Thomas Bach*. Aus Gründen der Übersichtlichkeit ist in der Abbildung nur ein kleiner Teil aller verbundenen Konzepte dargestellt. Es fehlen sowohl die zeitlichen als auch die geographischen Einteilungen der Einträge.

Während die interne Repräsentation der Konzepte des Netzes eindeutige URIs zur Definition der Konzepte und zur Identifikation der Verbindungen verwendet, sind in der Abbildung die Vorzugsnamen der jeweiligen Einträge dargestellt. Des Weiteren verfügen einige der dargestellten Konzepte neben ihren URIs und Vorzugsnamen über weitere, alternative Namen. So enthält der Eintrag zu *Thomas Bach* den alternative Namen *Thomas Bachs* und der Eintrag zu *Michael Bloomberg* den alternativen Namen *Michael R. Bloomberg*.

Selbst dieser kleine Ausschnitt aus dem TopicZoom Netz zeigt die fein geglie-

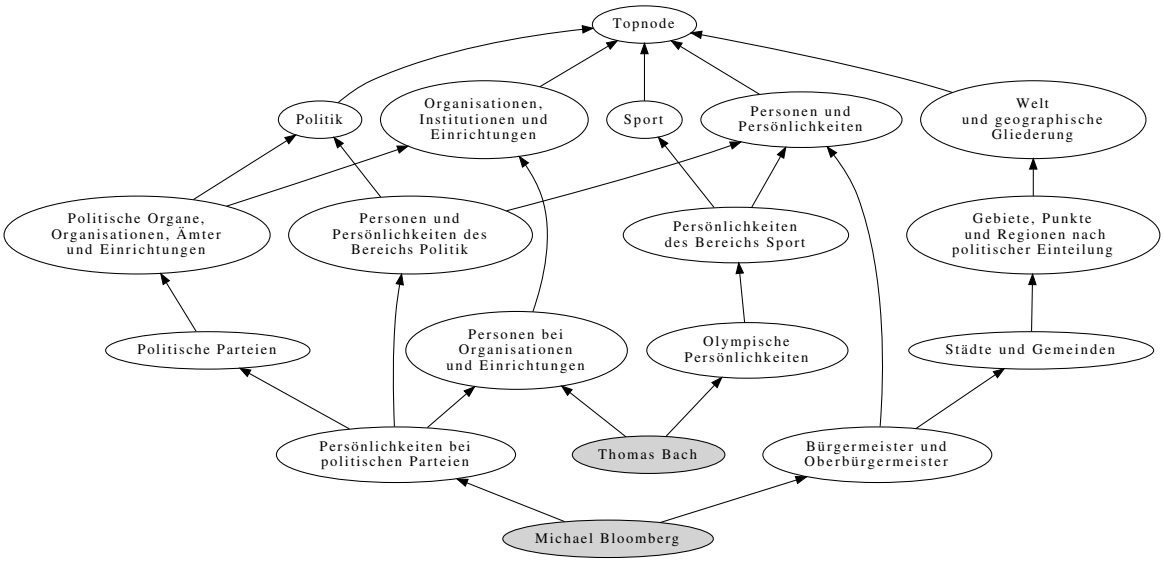


Abbildung 4.2.: Ausschnitt aus dem TopicZoom Netz zu den beiden benannten Entitäten *Michael Bloomberg* und *Thomas Bach*

derte Sortierung der Entitäten. Die beiden Personen des öffentlichen Lebens sind einerseits aufgrund ihrer unterschiedlichen Wirkungsfelder (*Sport* und *Politik*) klar voneinander abgegrenzt. Andererseits weisen beide sowohl als öffentliche Persönlichkeiten als auch als Mitglieder öffentlicher Einrichtungen eine Reihe thematischer Überlappungen auf. So sind beide Einträge den Themen *Personen und Persönlichkeiten* und *Personen bei Organisationen und Einrichtungen* direkt oder indirekt untergeordnet.

Wie alle EFGT-Netze weist das TopicZoom Netz einen einzigen Wurzelknoten auf. Dieser als *Topnode* bezeichnete Knoten stellt den Endpunkt jedes möglichen Weges durch das Netz dar. Alle Einträge im Netz sind letztendlich Unterthemen dieses Knotens. Anders als bei klassischen Taxonomien, die in verschiedenen (natur-) wissenschaftlichen Gebieten zum Einsatz kommen, sind EFGT-Netze weniger streng hierarchisch aufgebaut. Sie bilden keine baumartige Struktur ab und verschiedene Knoten verfügen über eine variable Anzahl von Eltern- und Kinderknoten. Auch wenn die Tiefe einzelner Knoten Hinweise auf ihre Spezifiziertheit geben[5], verfügen die Netze über keine starren Ebenen.

[5]Je tiefer ein Knoten im Netz liegt, um so spezifischer ist seine thematische Ausrichtung

4.4. Thesauri

Mit dem Begriff des *Thesaurus* ist eine weit verbreitete Form der Zusammenstellung von natürlichsprachlichen Begriffen bezeichnet. Thesauri sammeln verschiedene Konzepte eines bestimmten Wortschatzes und setzen die gesammelten Konzepte auf eine bestimmte Weise in Bezug zu anderen Konzepten dieses Wortschatzes. Thesauri sind unter anderem durch die beiden Standards ISO 25964–1[6] und ISO 25964–2[7][32] international standardisiert. Eine häufig genannte Standarddefinition ist nach der veralteten DIN 1463–1 Norm die folgende [10, S. 19]:

> Ein Thesaurus im Bereich der Information und Dokumentation ist eine geordnete Zusammenstellung von Konzepten und ihren (vorwiegend natürlichsprachigen) Bezeichnungen, die in einem Dokumentationsgebiet zum Indexieren, Speichern und Wiederauffinden dient. Er ist durch folgende Merkmale gekennzeichnet:
>
> 1. Konzepte und Bezeichnungen werden eindeutig aufeinander bezogen („terminologische Kontrolle“), indem
>   a) Synonyme möglichst vollständig erfasst werden,
>   b) Homonyme und Polyseme besonders gekennzeichnet werden,
>   c) für jeden Begriff eine Bezeichnung (Vorzugsbenennung, Begriffsnummer oder Notation) festgelegt wird, die den Begriff eindeutig vertritt,
> 2. Beziehungen zwischen Konzepten (repräsentiert durch ihre Bezeichnungen) werden dargestellt.

Der Standard stellt hauptsächlich zwei Bedingungen für die Einträge in einem Thesaurus auf. Zum einen sollen für jeden Begriff eindeutige Bezeichnungen festgelegt werden, die es ermöglichen, eindeutige Verbindungen herzustellen. Andererseits müssen die Beziehungen zu den anderen Konzepten des Thesaurus möglichst vollständig abgebildet werden. Hierbei dienen die Vorzugsbezeichnungen der Konzepte als eindeutige Identifikatoren.

Klassischerweise kodieren Thesauri noch unterschiedliche *Deskriptoren*. Diese dienen der genaueren Definition der Beziehungen der Konzepte untereinander.

[6] Information and documentation — Thesauri and interoperability with other vocabularies — Part 1: Thesauri for information retrieval

[7] Information and documentation — Thesauri and interoperability with other vocabularies — Part 2: Interoperability with other vocabularies

Sie geben darüber Auskunft *warum* zwei Konzepte zueinander in Beziehung stehen und definieren somit die Semantik der Beziehung. Es werden typischerweise folgende Beziehungen kodiert [10, S. 21]:

UF (engl. *Use for*) Weist von der Vorzugsbenennung auf die Synonyme der Synonymfamilie.

SYN (engl. *Synonym*) Stellt die umgekehrte Beziehung von *UF* dar.

BT (engl. *Broader term*) Verweist auf Hyperonyme (Oberbegriffe) des Begriffs.

NT (engl. *Narrower term*) Verweist auf Hyponyme (Unterbegriffe) des Begriffs.

RT (engl. *Related term*) Verweist auf generisch verwandte Konzepte.

TT (engl. *Top term*) Verweist auf einen maximal generalisierten Oberbegriff.

Die Deskriptoren sind sehr allgemein angelegt. Somit können die genauen Bedeutungen der Beziehungen von Thesaurus zu Thesaurus und auch von Begriff zu Begriff innerhalb desselben Thesaurus variieren. Es ist zum Beispiel nicht festgelegt, welche genaue Beziehung BT festschreibt — je nach verwendeten Thesaurus kann eine *ist ein* oder eine *liegt in* Beziehung kodiert sein. Im Allgemeinen wird durch die BT und NT Beziehungen

> [...] eine irgendwie geartete hierarchische Verwandtschaft abgebildet[10, S. 21]

Thesauri stellen eine einfache Möglichkeit dar, einen Wortschatz oder auch allgemeiner Wissen strukturiert abzuspeichern. Sie werden meist dazu eingesetzt, domänenspezifisches Wissen oder den Wortschatz einer bestimmten Sprache zu kodieren. Es gibt eine Vielzahl frei verfügbarer Thesauri in unterschiedlichen Sprachen und zu unterschiedlichen Themen. Gerade die breite Verfügbarkeit von Thesauri macht sie zu idealen Wissensbasen, um Texte semantisch zu indexieren.

Allerdings weisen die meisten Thesauri eine Reihe von Defekten auf [10, S. 24]. Einerseits richten sie sich oft direkt an menschliche Benutzer. Es werden oft zwar Benennungsvarianten, nicht aber Flexions- und Derivationsformen erfasst. Andererseits sind die meisten verfügbaren Thesauri entweder sehr domänenspezifisch aufgebaut oder die Beziehungen der Konzepte untereinander nur sehr grob abgebildet. Viele dieser Probleme betreffen allerdings nicht nur Thesauri. Vielmehr bestehen diese Probleme bei einer Vielzahl von Formalismen zur Repräsentation von Wissen.

4.5. Ontologien

Ontologien sind ursprünglich ein Begriff aus der Philosophie. Er bezeichnet eine Disziplin aus der theoretischen Philosophie, die sich mit dem *Seinenden* und der Wirklichkeit beschäftigt. In dem Gebiet der *AI-Forschung* und allgemeiner in den Computerwissenschaften bezeichnen Ontologien einen Formalismus zur Repräsentation von Wissen. Sie werden oft als ein Tupel $\langle D, R \rangle$ aus einer Menge D und einer Menge von Relationen R auf D repräsentiert [27, 26, 22, 28]. Dabei bezeichnet D eine nicht weiter spezifizierte Menge aus Entitäten bzw. *concepts (Begriffen)* [28], das sogenannte *Universe of discourse* (vgl. Kapitel 1.5). In dieser Arbeit werden ausschließlich Ontologien zur Wissensrepräsentation betrachtet.

Die Beziehungen der Konzepte in Ontologien untereinander spannen ein Netz auf. Dieses Netz unterliegt im Gegensatz zu Thesauri keiner strikten hierarchischen Ordnung. Sie können einen Wurzelknoten besitzen, weisen aber im Gegensatz zu den EFGT-Netzen im Allgemeinen keinen solchen zentralen Wurzelknoten auf. Die Verbindungen in einer Ontologie können Zyklen aufweisen. Die Art der Beziehungen unter den Objekten sind bei Ontologien frei definierbar — sie sind somit theoretisch dazu in der Lage, beliebige Zusammenhänge zwischen Objekten zu modellieren.

Im Allgemeinen bestehen Ontologien aus einer Menge von Entitäten und deren Beziehungen untereinander. Hierbei wird oft noch genauer zwischen *Klassen*, *Instanzen*, *Prädikaten* und *Axiomen* unterschieden.

Klassen definieren eindeutig die vorhandenen abstrakten Konzepte innerhalb der Ontologie. Die Verbindungen der Objekte untereinander bilden ein hierarchisches Netz aus Ober- und Unterbegriffen. Sie können als Klassen in der objektorientierten Programmierung aufgefasst werden. Klassen realisieren abstrakte, mengenartige Konzepte wie zum Beispiel *Menschen*, *Tiere* oder *Büromöbel*.

Instanzen bezeichnen die einzelnen Einträge (die *konkreten Objekte*) der Ontologie. Sie sind gleichermaßen mit Begriffen und anderen Instanzen der Ontologie verbunden. Sie stellen einzelne Realisierungen von Klassen dar. Sie können als Klasseninstanzen in der objektorientierten Programmierung angesehen werden und bilden konkrete Realisierungen einzelner Konzepte. Instanzen sind meist Konzepte, die Individuen und konkrete Realisierungen von Objekten, wie zum Beispiel die Person *Florian Fink* oder *Florians Schreibtischstuhl* modellieren.

Prädikate bezeichnen Beziehungen unter den Instanzen und Begriffen der Ontologie. Es kommt dabei vor allem auf das verwendete Ontologieschema an, wel-

che Formen von Beziehungen zwischen den Objekten zugelassen beziehungsweise verboten sind. Im Allgemeinen sind der Definition von Beziehungen der Begriffe und Instanzen untereinander allerdings keine Grenzen gesetzt. Wichtig ist dabei auch, dass mit Ontologien — anders als zum Beispiel mit EFGT-Netzen — nicht nur allgemeine Verbindungen, sondern durch Namen von Relationen auch Typen von Verbindungen ausgezeichnet werden können. Im Allgemeinen dienen die Prädikate in Ontologien dazu die vielen Beziehungen der Klassen und Instanzen untereinander zu beschreiben und bilden die Menge der Relationen in der Ontologie.

Axiome bzw. Folgerungen bezeichnen die aus den Relationen ableitbaren Beziehungen, die nicht direkt in den Relationen abgebildet sind. Sie können als das Wissen angesehen werden, das aus den impliziten Verbindungen geschlussfolgert werden kann. Die aus den Axiomen ableitbaren Beziehungen bilden wiederum neue abgeleitete Prädikate, die nicht explizit ausgezeichnet sind, sondern implizit abgeleitet werden können.

Beispiel 4.5.1. Eine einfache Ontologie enthalte unter anderem die beiden miteinander verbundenen Klassen *Komponist* und *Mensch* sowie die beiden miteinander verbundenen Instanzen *Ludwig van Beethoven* und *Franz Joseph Haydn*. Zur näheren Beschreibung der beiden Instanzen in der Ontologie sind sie zusätzlich noch mit der Klasse *Komponist* verbunden.

Aus diesen Verbindungen kann nun einerseits direkt herausgelesen werden, dass die beiden Instanzen Komponisten waren, andererseits kann axiomatisch abgeleitet werden, dass die beiden Instanzen dann auch Menschen sein müssen, da der Begriff *Komponist* mit dem Begriff *Mensch* verbunden ist.

Ontologien bilden das in ihnen abgespeicherte Wissen vor allem über die Verbindungen der Konzepte und Instanzen untereinander ab. Anders als bei Thesauri kann hierbei aber die genaue Art der Verbindung spezifiziert werden. Die Verbindungen der Konzepte und Instanzen der Ontologie haben Namen, die die Semantik der Bedeutung festlegen.

Die elementaren Einheiten der Ontologie sind einzelne Fakten. Diese Elemente werden dabei oft auch als *Tripel* bezeichnet, da die Relationen der Ontologie in den verschiedenen Implementierungen als dreielementige Tupel repräsentiert werden. Die Tripel der Relationsmenge einer Ontologie bestehen immer aus zwei miteinander verbundenen Instanzen oder Konzepten und einem zusätzlichen Element, welches den Namen der Verbindung spezifiziert. Dieses zusätzliche Element definiert die Bedeutung der Verbindung und somit die Semantik des dargestellten Faktums. Oft ist es ebenso wie die anderen beiden Elemente ein eindeutiges, meist über eine URI spezifiziertes Objekt, welches selbst wiederum mit anderen Teilen

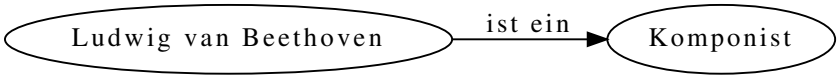


Abbildung 4.3.: Graphische Darstellung eines einfachen Faktentripel

der Ontologie in Verbindung steht. Die drei Elemente der Tripel werden oft als *Subjekt*, *Prädikat* und *Objekt* bezeichnet.

Diese Namensgebung kommt daher, dass man Faktentripel als einen einfachen Satz aus Subjekt, Objekt und Prädikat verstehen kann, wobei im Allgemeinen jeweils das erste Element das Subjekt, das zweite Element das Prädikat und das dritte Element das Objekt darstellt. Die Fakten der Relation können so auch als einfache Sätze interpretiert werden, die Aussagen über die Objekte der Ontologie tätigen. Einfache Faktentripel werden auch oft zur Verdeutlichung graphisch dargestellt, wobei Subjekt und Objekt des Tripels als zwei Knoten dargestellt werden, die mit einer, durch das Prädikat des Tripels markierten Verbindung, vom Subjekt hin zum Objekt verbunden sind (siehe Abbildung 4.3).

Beispiel 4.5.2. Das Faktum aus dem vorangegangenen Beispiel 4.5.1, dass *Ludwig van Beethoven ein Komponist ist* könnte in der Tripelschreibweise der Ontologie als $\langle$*Ludwig van Beethoven*, *ist ein*, *Komponist*$\rangle$ dargestellt werden. *Ludwig van Beethoven* bezeichnet dabei das Subjekt, *Komponist*, das Objekt und *ist ein* das Prädikat.

Aus diesen drei Teilen kann man nun den einfachen Satz *Ludwig van Beethoven ist ein Komponist* ableiten, der genau diese Tatsache in einem natürlichsprachlichen Satz ausdrückt.

Es wurde eine ganze Reihe von Auszeichnungssprachen zur Beschreibung von Ontologien entwickelt. Diese Auszeichnungssprachen formalisieren und unterstützen die Erstellung von Ontologien, indem sie unter anderem weitere Regeln für die Verbindungen zwischen Konzepten und Instanzen festschreiben. Basierend auf diesen Auszeichnungssprachen, wurden eine Reihe von Werkzeugen entwickelt, die die manuelle Erstellung von Ontologien erleichtern sollen [10, 43].

Ebenso wie bei den Thesauri gibt es eine Vielzahl unterschiedlicher, frei verfügbarer Ontologien, die an verschiedenen Stellen entwickelt und verbessert werden.

4.5.1. Thesauri und Ontologien

Wie es bei den vorangegangenen Erläuterungen zu den Ontologien bereits angeklungen ist, weisen Thesauri und Ontologien bestimmte Ähnlichkeiten, aber auch bestimmte Unterschiede auf.

Neben der Unterscheidung von Instanzen und Konzepten, können in Ontologien die Beziehungen zwischen den Konzepten und Instanzen frei gewählt werden.

Die starre Definition der Deskriptoren in Thesauri ist weit weniger flexibel und erlaubt nur einige wenige Typen von Verbindungen. Es können vor allem strikt hierarchische Strukturen mit ihnen dargestellt werden. Die Elemente der Ontologie dagegen sind viel flexibler und es können eine Vielzahl von Verbindungen und Eigenschaften zwischen den Elementen ausgezeichnet werden.

Beiden Ansätzen zur Darstellung von Wissen ist dagegen gemein, dass beide unterschiedliche Begriffe zueinander in Beziehung setzen und auch beide ein mehr oder weniger stark ausgeprägtes Konzept von *benannten Relationen* beinhalten. Das soll heißen, dass die Art der Verbindung die Semantik der Verbindung kodiert, auch wenn die Flexibilität bei der Darstellung von unterschiedlichen Verbindungen bei Thesauri deutlich geringer ist.

Es ist somit durchaus möglich Thesauri ebenso wie Ontologien durch eine Menge aus Faktentripeln zu repräsentieren, die jeweils aus zwei miteinander verbundenen Begriffen und einem zusätzlichen Namen für die Beziehung bestehen. Der Name dieser Beziehung entspricht dann im Falle von Thesauri einfach dem entsprechenden Deskriptor der Verbindung. Dies ermöglicht es, Ontologien und Thesauri weitestgehend gleich zu behandeln. Wegen der mangelnden Flexibilität von Thesauri können mit ihnen allerdings nur eingeschränkte Wissensbasen dargestellt werden. Es ist in Thesauri unter anderem nicht möglich verschiedene Eigenschaften der Deskriptoren festzulegen oder auf einfache Weise neue Deskriptoren zu erzeugen

Im weitergehenden Verlauf dieser Arbeit wird aus den hier dargestellten Gründen zwischen Thesauri und Ontologien nicht weiter unterschieden. Es wird ebenso nicht weiter zwischen Instanzen und Klassen unterschieden, da diese Unterscheidung insbesondere nur für Ontologien eine Rolle spielt und vor allem zum Aufbau der Wissensressource nötig ist, nicht aber bei der semantischen Indexierung. Klassen und Instanzen können einheitlich als Konzepte angesehen werden, ihre Unterschiede liegen vor allem in den Beziehungen untereinander. Die verschiedenen Prädikate in Ontologien entsprechen genau den Relationen der Relationsmenge der expliziten Wissensressourcen aus Definition 1.5.1. Auch können die Axiome in Ontologien weitestgehend ignoriert werden, da diese leicht als weitere, aus den bestehenden Relationen der Ontologie abgeleitete[8], Relationen betrachtet werden können.

Beide Ansätze zur Wissensrepräsentation können als Wissensressourcen zur semantischen Indexierung verwendet werden. Man kann dabei Thesauri als eine stark eingeschränkte Ontologie ansehen. Dies erleichtert zum einen die weiteren Betrachtungen zur semantischen Indexierung und andererseits hat die vereinheit-

[8]Meist werden hierzu die symmetrischen bzw. transitiven Eigenschaften von Relationen betrachtet.

lichte Behandlung von Ontologien und Thesauri den weiteren Vorteil, dass die zur semantischen Indexierung verwendbaren Wissensressourcen gleichermaßen auf die Vielzahl von frei verfügbaren Thesauri und Ontologien zugreifen können, ohne zwischen den beiden Darstellungsarten unterscheiden zu müssen.

4.5.2. Ontologie „Erster Weltkrieg“

Die zweite im Rahmen dieser Arbeit zu Testzwecken verwendete Wissensressource ist die *Ontologie „Erster Weltkriegs“*. Diese ist eine domänenspezifische Ontologie, in der Wissen über wichtige Persönlichkeiten, Orte und Ereignisse des ersten Weltkriegs in den drei Sprachen russisch, deutsch und englisch in einer Ontologie abgespeichert ist [44]. Sie wird im weiteren Verlauf der Arbeit dazu dienen, verschiedene Aspekte der semantischen Indexierung mit Ontologien darzustellen.

Konzepte der Ontologie

Die Ontologie verbindet historische Ereignisse des ersten Weltkriegs mit den teilnehmenden Ländern und Personen, sowie mit ihren jeweiligen Standorten. Sie umfasst Einträge zu wichtigen historischen Persönlichkeiten und geopolitische Einträge wie Länder, Städte und Flüsse. Die Ontologie ist implizit typisiert und die Haupteinträge der Ontologie stellen Konzepte dar, die jeweils einem der drei Typen *PERSON*, *GEO* oder *EVENT* zugeordnet sind. Dabei steht *PERSON* für Einträge zu wichtigen militärischen und politischen Persönlichkeiten dieser Zeit, *GEO* für geopolitische Einträge jeder Art und *EVENT* für wichtige Ereignisse während der Zeit des ersten Weltkriegs. Um die Darstellung zu vereinheitlichen, wird die ursprüngliche Typisierung in der Ontologie „Erster Weltkrieg“ aufgebrochen, indem den entsprechenden Konzepten über ein neu eingeführtes Prädikat *Type* jeweils eines der drei — ebenfalls neu eingeführten Konzepte — *PERSON*, *GEO* und *EVENT* zugeordnet wird.

Die Ontologie verfügt über eine Reihe von Konzepten, die natürlichsprachliche Ausdrücke in drei unterschiedlichen Sprachen Russisch, Englisch und Deutsch darstellen. Diese natürlichsprachlichen Ausdrücke werden über das Prädikat *Name* mit anderen Konzepten der Ontologie verbunden, wobei in der Darstellung in dieser Arbeit nicht weiter zwischen den Sprachen der Ausdrücke unterschieden wird.

Die Konzepte der Ontologie weisen eine Vielzahl weiterer Metadaten auf. Einträge zu Personen umfassen Todes- und Geburtsdaten, verschiedene Berufe und ihren gesellschaftlichen Geltungsbereich. Ereignisse sind in eine Hierarchie von Subtypen eingeteilt, die die Ereignisse je nach Typ feiner spezifizieren. Wie Einträge zu Personen verfügen sie auch über weitere zeitliche Angaben zum jeweiligen

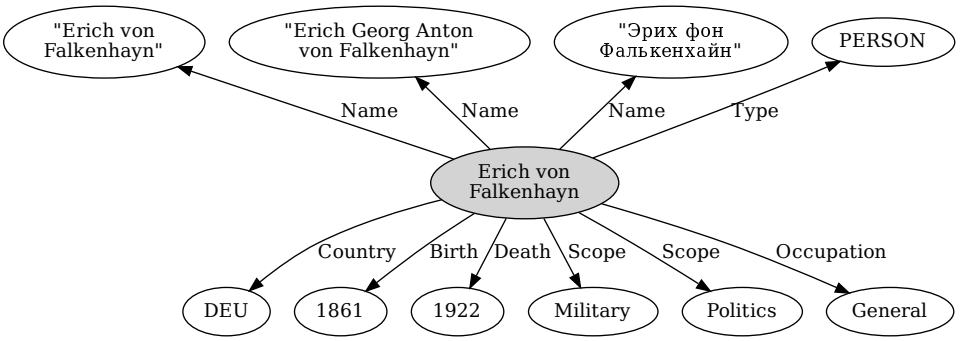


Abbildung 4.4.: Ausschnitt aus der Ontologie „Erster Weltkrieg“ zum Eintrag zur Person *Erich von Falkenhayn* (grau).

Ereignis. Ortsangaben sind einerseits in eine geopolitische Hierarchie einsortiert, die Städte und Flüsse ihren entsprechenden Ländern zuteilt und andererseits in eine Hierarchie von Subtypen eingeteilt.

Relationen der Ontologie

Die Ontologie enthält neben den beiden bereits erwähnten Prädikaten *Type* und *Name* zwölf weitere Prädikate, mit denen die Konzepte der Ontologie untereinander verknüpft sind. Es bestehen dabei keinerlei Einschränkungen bezüglich der Anzahl der erlaubten Verbindungen. Verschiedene Konzepte können beliebig oft auch unter demselben Prädikat mit anderen Konzepten verknüpft sein.

Country Diese Relation ordnet einerseits Konzepte des Typs *PERSON* ihren entsprechenden Ländern zu, wobei die zugeordneten Länder eher die politische oder militärische Zugehörigkeit angeben als das Geburtsland der Person. Andererseits werden geographische Konzepte wie Flüsse, Städte oder Gebirge ihren geopolitischen Ländern zugeordnet, wobei einzelne Konzepte durchaus mehrere Länder über diese Relation referenzieren können.

Death und Birth Diese beiden Relationen spezifizieren Geburts- und Todesdaten von Personen. Sie verbinden Konzepte des Typs *PERSON* mit externen Konzepten, die verschiedene Jahreszahlen und Datumsangaben repräsentieren. Zur einfacheren Darstellung der Ontologie werden nur Jahreszahlen betrachtet. In Wirklichkeit referenzieren die Relationen *Birth* und *Death* meist volle Datumsangaben.

Party Diese Relation definiert die Parteizugehörigkeit von Personen. Sie verbindet Konzepte des Typs *PERSON* mit Metakonzepten der Ontologie, die die verschiedenen politischen Parteien des ersten Weltkriegs repräsentieren.

Scope und Occupation Diese beiden Relationen definieren den Geltungsbereich (Scope) und den Beruf und Titel (Occupation) einzelner Personeneinträge der Ontologie. Sie verbinden Konzepte des Typs *PERSON* mit Metakonzepten, die verschiedene Geltungsbereiche und Berufsbezeichnungen umfassen.

CountriesInvolved und PersonsInvolved Diese Relationen verbinden Konzepte des Typs *EVENT* mit Konzepten des Typs *GEO* (CountriesInvolved) beziehungsweise mit Konzepten des Typs *PERSON* (PersonsInvolved). Über diese Verbindungen können die wichtigsten Personen und Länder spezifiziert werden, die für bestimmte Ereignisse eine Rolle spielen.

Date Diese Relation definiert den Zeitpunkt bestimmter Ereignisse. Sie Verbindet Konzepte des Typs *EVENT* mit Zeitpunkten, die als Metakonzepte implizit in der Ontologie vorliegen. Wie schon bei den Relationen *Birth* und *Death* erwähnt werden hier nur Jahreszahlen betrachtet, obwohl die tatsächliche Ontologie volle Datumsangaben verwendet.

EventType Diese Relation dient einer Klassifikation von Ereignissen. Sie Verbindet Konzepte des Typs *EVENT* mit Metakonzepten, die unterschiedliche Typen von Ereignissen repräsentieren. Darunter zum Beispiel *Battle* (Schlachten), *Sinking* (Versenkung von Schiffen) und *Treaty* (Abkommen jeglicher Art).

PartOf Diese Relation verbindet Konzepte des Typs *EVENT* miteinander. Sie dient einer hierarchischen Einordnung von Ereignissen. So sind verschiedenen Schlachten der Ontologie Teile größerer militärischer Kampagnen und Offensiven zugeordnet.

Place Diese Relation verbindet Konzepte des Typs *EVENT* mit Konzepten des Typs *GEO*. Über diese Relationen kann der Ort von Ereignissen spezifiziert werden, an dem diese stattfanden.

Type Diese Relation ist nicht expliziter Teil der Ontologie. Sie kann aber beim Parsen der Ontologie mit erzeugt werden, um die implizite Typisierung der Konzepte explizit in die Wissensressource mit aufzunehmen. Neben den bestehenden, impliziten Typen *PERSON*, *EVENT* und *GEO* werden auch Datumsangaben mit dem

Typen *DATE* und Berufe und Geltungsbereiche mit den Typen *OCCUPATION* bzw. *SCOPE* markiert. Diese explizite Typisierung ermöglicht es bei einer späteren semantischen Indexierung auf die Typen der Konzepte über diese Relation zuzugreifen, um so auch auf Ereignisse, Orte, Personen und Daten der Ontologie zugreifen zu können.

Beispiel 4.5.3. Hier soll die Ontologie kurz anhand des Eintrags zur *Schlacht um Verdun* vorgestellt werden. Dieses Beispiel wird dann im weiteren Verlauf der Arbeit zur Verdeutlichung von unterschiedlichen Sachverhalten dienen.

Der für dieses Beispiel zentrale Eintrag in der Ontologie ist der Eintrag zum Ereignis *Schlacht um Verdun*. Neben einer Reihe weiterer Metadaten verweist der Eintrag auf eine Reihe von Personen und Ländern, die an dieser Schlacht beteiligt waren. Darunter sind unter anderem die Personen *Philippe Pétain* und *Erich von Falkenhayn* und die beteiligten Länder *Frankreich* und *Deutschland*. Der Eintrag verweist ebenso auf das Konzept der Stadt Verdun, bei der das Ereignis stattfand.

Der Eintrag vom Type *GEO* zur Stadt *Verdun* enthält seinerseits wiederum eine Verbindung zum Eintrag Frankreich. Der Eintrag enthält auch eine Reihe von alternativen Schreibweisen für die Stadt, die diesem über das Prädikat *Name* zugeordnet werden. Darunter *Верден*, *Wirten* und *Verden*. Alle diese Einträge sind alternative, teils historische Schreibweisen für die Stadt *Verdun*. Sie helfen die Begriffe in Texten in unterschiedlichen Sprachen und mit höherem Recall zu identifizieren.

Neben den Einträgen zu *Frankreich* und *Deutschland*, die wiederum ähnliche Fakten enthalten wie der Eintrag zur Stadt *Verdun*, enthält die Ontologie auch Einträge für alle wichtigen historischen und politischen Persönlichkeiten, die an der *Schlacht um Verdun* beteiligten waren. Die Einträge zu diesen Personen enthalten wiederum neben einigen Metadaten Verbindungen zu ihren zugeordneten Ländern und auch weitere alternative Schreibweisen ihrer Namen und Titel.

Abbildung 4.5 stellt nun abschließend das vereinfachte Netz dar, das der Eintrag zur *Schlacht um Verdun* mit all seinen Verbindungen aufspannt. Aus Gründen der Übersichtlichkeit wurde auch auf die Darstellung der expliziten Typisierung der Konzepte verzichtet. In der tatsächlichen Ontologie ist die *Schlacht um Verdun* mit dem Typ *EVENT* verknüpft, die jeweiligen Personen und Orte jeweils mit den Konzepten *PERSON* bzw. *GEO*.

Einerseits zeigt diese Abbildung, wie komplex selbst einfache Fakten eines begrenzten Ereignisses untereinander vernetzt sind. Andererseits kann man bei der Betrachtung der Abbildung sehen, wie Ontologien in der Lage dazu sind, komplexe Zusammenhänge auf Basis einfacher Verbindungen, gleichermaßen dem menschlichen Benutzer und einer Maschine verständlich zu machen.

Es bleibt hier noch anzumerken, dass die wirklichen Einträge zu allen mit der *Schlacht um Verdun* verbundenen Konzepte in der Ontologie über den ersten Welt-

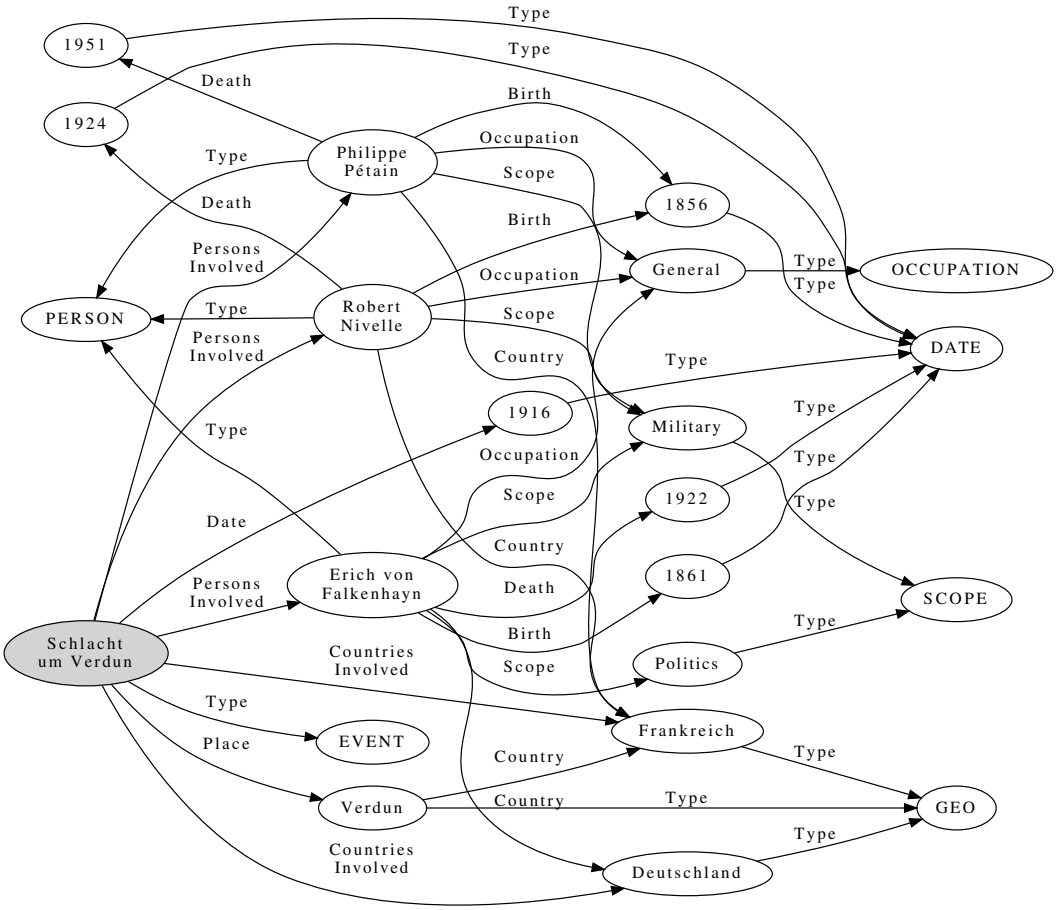


Abbildung 4.5.: Ausschnitt aus der Ontologie „Erster Weltkrieg“ zum Eintrag des Ereignisses *Schlacht um Verdun*. Aus Gründen der Übersichtlichkeit sind nicht alle Verbindungen der Originaleinträge, insbesondere die Namen, dargestellt.

krieg in Wirklichkeit ein sehr viel komplexeres und größeres Netz aufspannen, das nicht mehr so einfach auf einer Buchseite dargestellt werden kann.

4.6. Auszeichnungssprachen für Ontologien

Wie bereits erwähnt, wurden eine Vielzahl unterschiedlicher Auszeichnungssprachen zur Erstellung von Ontologien entwickelt. Diese versuchen einerseits die Erstellung von Ontologien zu erleichtern und andererseits die Repräsentation von Ontologien zu vereinheitlichen. Hier werden nun kurz zwei weit verbreitete Auszeichnungssprachen vorgestellt, die jeweils aufeinander aufbauen. Ein weiterer Standard zur Auszeichnung von Ontologien wird zusätzlich noch kurz erwähnt. Die folgenden Ausführungen sollen klar machen wie Wissen auf sehr kompakte und einfache Weise auf der Basis von Ontologien festgeschrieben werden kann. Diese auf eine solche Weise dargestellten Wissensressourcen können dann die eigentliche Basis für die semantische Indexierung darstellen, indem die in der Wissensressource dargestellten Fakten vor dem Indexierungsprozess in entsprechende Datenstrukturen umgewandelt werden.

4.6.1. Resource Description Framework

Eine verbreitete Auszeichnungssprache zur Erstellung von Ontologien ist das XML-basierte *Resource Description Framework (RDF)*. Es wurde ursprünglich 1999 entwickelt, um Metadaten von XML Ressourcen zu beschreiben. Dieses Gerüst wurde dann 2004 so erweitert, dass nicht nur Metadaten von Dokumenten, sondern auch Beziehungen zwischen Objekten dargestellt werden konnten [57]. RDF wurde vom *World Wide Web Consortium (W3C)*[9] entwickelt und von ihnen als Standard zur Wissensrepräsentation empfohlen.

Über die Zeit sind für RDF eine Vielzahl von Anwendungsprogrammen und Programmbibliotheken entstanden, um die Handhabung der Dokumente und den Zugriff auf die in ihnen gespeicherte Information zu erleichtern. Ebenso wird RDF in einer ganzen Reihe kommerzieller oder quelloffener Anwendungen verwendet. Ein bekanntes Beispiel hierzu bietet das *RSS (RDF Site Summary)* Format, welches zum Abonnement von Nachrichten eingesetzt wird und auf dem RDF Format basiert.

Die grundlegende Form in RDF Dokumenten um Fakten darzustellen sind die, in Kapitel 4.5 beschriebenen, Faktentripel. Ebenso wie bei Ontologien, spannen die Fakten des RDF Dokuments einen gerichteten und gelabelten Graphen auf, der das in der Wissensbasis dargestellte Wissen repräsentiert.

[9] `http://www.w3.org`

Die Auszeichnungssprache, die im RDF Framework zur Definition von Tripeln verwendet wird, ist XML-basiert. RDF verwendet für seine Deklarationen den RDF-spezifischen XML Namensraum `rdf`. Alle Subjekt-Objekt-Prädikat Fakten werden in RDF als eigene XML Knoten repräsentiert. In der Sprechweise von RDF, die sich noch aus der ursprünglichen Aufgabe der Metadatendefinition ableiten lassen, werden die drei Elemente der Tripel als *ressource* (Subjekt), *objekt* (Objekt) und *property* (Prädikat) bezeichnet.

Alle drei Elemente der einzelnen Faktentripel müssen laut Spezifikation über eine URI eindeutig definiert sein [43, S. 36]. Diese Forderung soll einerseits kontextabhängige Mehrdeutigkeiten, Ambiguitäten innerhalb von Sprachen oder über verschiedene Sprachen hinweg und Ambiguitäten von Namen verhindern und andererseits die Wiederverwendbarkeit einzelner Elemente fördern.

Aus dieser Forderung folgt direkt auch, dass nun nicht mehr nur die Konzepte, also die Subjekte und Objekte mit eindeutigen URIs identifiziert sein müssen, sondern auch die Prädikate der Verbindungen.

Beispiel 4.6.1. Ein Beispiel für die eben erwähnte Kontextabhängigkeit von Begriffen stellt der Begriff der hier behandelten Ontologie selbst dar. In einem philosophischen Kontext verweist *Ontologie* auf eine Disziplin der Philosophie, wohingegen die in diesem Text bevorzugte Bedeutung des Begriffs, die hier beschriebene Form der Wissensrepräsentation verwendet (vgl. Abschnitt 4.5 auf Seite 75 weiter vorne). Beide Begriffe müssen in RDF Dokumenten eindeutig über unterschiedliche URIs identifiziert sein. So könnten die beiden Begriffe über ihre URIs `http://www.example.org/Ontologie(Philosophie)` und `http://www.example.org/Ontologie(Computerwissenschaften)` eindeutig unterschieden werden.

Aus der Forderung der Eindeutigkeit der verwendeten Objekte folgt insbesondere auch, dass alle Objekte in RDF Dokumenten *global* über alle Dokumente definiert sind. Objekte notieren somit immer eindeutige Entitäten und sind eineindeutig in *allen* RDF Dokumenten. Nur triviale Objekte, wie etwa Datumsangaben oder Identifikationsnummern aller Art, deren Interpretation in jedem beliebigen Kontext eindeutig ist, können einfacher auch über ihre jeweiligen String- oder Zahlenwerte notiert werden. Alle anderen Objekte dagegen müssen eindeutig spezifiziert sein.

Aufgrund der Tatsache, dass die meisten Objekte eines RDF Dokuments eindeutig über URIs spezifiziert sein müssen und der Tatsache, dass XML dazu tendiert übermäßig auszuzeichnen, sind die in einem Dokument ausgezeichneten Fakten schwer nachzuvollziehen. Nichtsdestotrotz bietet RDF eine flexible Möglichkeit, nicht nur einfache Fakten zu sammeln und auszuzeichnen, sondern auch die Möglichkeit verschiedene, RDF-konforme Faktensammlungen einfach zu kombinieren.

4.6.2. Resource Description Framework Schema

Um zusätzlich zu den einfache Fakten einer Wissensressource noch axiomatische Ableitungen, wie sie aus den Ontologien bekannt sind, darstellen zu können, wurde zusätzlich zum RDF Format noch ein sogenanntes *Resource Description Framework Schema (RDFS)* eingeführt. Diese Auszeichnungssprache definiert zusätzlich zum einfachen XML Format noch eine zugehörige Schemasprache.

Anders als die aus den XML Formaten bekannten Schemadefinitionen, wie z.B. DTD oder W3C XML [60, 61, 62, 34], dient RDFS nicht allein zur Definition der *Syntax* von Dokumenten, sondern vielmehr der Definition der *Semantik* in RDF Dokumenten. Mit RDFS ist es möglich, sowohl die Arten und Eigenschaften der Objekte auszuzeichnen als auch die Relationen, die zwischen den Objekten bestehen, zu definieren und ebenso Eigenschaften für diese zu definieren. Mit solchen Schemas können nun aus den expliziten Fakten weitere implizite Fakten abgeleitet werden [10, S. 35].

Im einfachen RDF Format werden Fakten als einfache Subjekt Objekt Prädikat Tripel definiert. Dabei werden zwei Individuen über ein Prädikat in Beziehung gesetzt. Zur weiteren Spezifizierung von einfachen Individuen können im einfachen RDF Format individuelle Typen vergeben werden. Diese Typisierung kann einerseits als übergeordnete hierarchische Einordnung angesehen werden, zum anderen auch als eine spezielle Form des *„ist ein"* Prädikats, welches aus der objektorientierten Programmierung auch als Vererbung bekannt ist. Durch diese Typisierung können Individuen mit ähnlichen Eigenschaften einfach zusammen gruppiert werden.

RDFS erweitert nun diese einfache Form der Typisierung um das Konzept der RDFS-Klassen. Ziel hierbei ist, sowohl Individuen bestimmten Klassen zuzuordnen als auch bestimmte Regeln von Zugehörigkeiten von Individuen und Klassen anzugeben. Darüber ist es nun möglich, die einfachen, ansonsten nicht weiter unterscheidbaren, Individuenkonstanten der Ontologie mit zusätzlicher Logik zu versehen. Dies geschieht sowohl über die *Aggregation* als auch über die *Vererbung* von Klassen. Die Aggregation definiert ganz einfach ein *„hat ein"* Prädikat, wohingegen die Vererbung das bereits erwähnte *„ist ein"* Prädikat darstellt.

Klassen können also über eine spezielle Menge von anderen Klassen verfügen und somit können Individuen, die einer solchen Klasse angehören, immer in Beziehung zu anderen Individuen anderer Klassen stehen, mit denen sie aggregiert sind. Des Weiteren können Klassen und Individuen in diesem System von anderen Klassen erben. Dies bedeutet dann, dass alle Klassen und Individuen der erbenden Klasse auch als allgemeinere Instanzen der vererbenden Klasse angesehen werden können und rekursiv deren Eigenschaften erben.

Beispiel 4.6.2. Als einfaches Beispiel stelle man sich die hypothetische Klasse *Buch*

vor. Diese Klasse hätte zum Beispiel immer mindestens einen Autor, einen Verlag und einen Titel. Sowohl der Autor als auch der Verleger wären in diesem Fall Instanzen bzw. Individuen von der Klasse Autor oder Verlag. Man müsste für ein bestimmtes Individuum der Klasse *Buch* somit immer ein Individuum, welches von der Klasse *Autor* erbt und ein Individuum, welches von der Klasse *Verlag* erbt, angeben.

In einer solchen Ontologie könnten weitere Klassen definiert sein, von denen die Klassen *Buch*, *Autor* und *Verlag* wiederum erben. So würde die Klasse Autor noch von der Klasse *Person* erben, die Klasse *Buch* unter anderem von einer Klasse *Druckerzeugnisse* und die Klasse *Verlag* von einer Klasse *Unternehmen*. Von diesen Klassen aus könnten natürlich noch weitere Vererbungshierarchien zu anderen Klassen bestehen. Auch könnten diese Klassen selbst wieder andere Klassen über Aggregation an sich binden. Es ist somit vorstellbar, das ein Verlag unter anderem auch über ein Menge von Instanzen der Klasse *Person* verfügen kann oder muss.

Dieses Beispiel sollte nun verdeutlicht haben, wie sich über die einfachen *„ist ein"* und *„hat ein"* Prädikate bzw. die Aggregation und die Vererbung, Weltwissen in einer allgemeinen Form modellieren lässt. RDF-Schemata erlauben es nun Klassen zu definieren und die unterschiedlichen Beziehungen der Klassen untereinander zu spezifizieren. Die auf diese Weise in den Daten eingebetteten Informationen werden dann bei der Erzeugung der, zur semantischen Indexierung nötigen Datenstrukturen, herangezogen um axiomatische Ableitungen explizit in die Daten einzufügen. Hierbei können vor allem transitive *„ist ein"* Ableitungen durch Hüllenbildungen (vgl. Kapitel 1.1.2) automatisch erzeugt werden, wohingegen die Aggregation mittels *„hat ein"* explizit in den Daten ausgezeichnet sein muss.

4.6.3. Weitere Auszeichnungssprachen

Neben RDFS, hat sich die sogenannte *Web Onotlogy Language (OWL)* als ein weiterer Formalismus durchgesetzt. Dieser Formalismus baut ebenso wie RDFS auf RDF auf und bietet weitere Hilfsmittel zur Erstellung von Ontologien. Darunter verschiedene mengentheoretische Operationen wie Vereinigung und Durchschnitt, sowie die Möglichkeit, verschiedene Eigenschaften wie Symmetrie oder Transitivität einzelner Relationen (Prädikate) zu definieren. Diese Auszeichnungssprache wird hier nicht weiter beschrieben. Sie wird in [43] und [10] ausführlich behandelt.

Um die Unübersichtlichkeit der in XML ausgezeichnenten Wissensressourcen zu verringern hat sich in den letzten Jahren eine weitere, nicht XML-basierte Auszeichnungssprache entwickelt [4]. Diese als *Terse RDF Triple Language (Turtle)* bezeichnete Auszeichnungssprache, bietet die Möglichkeit die Fakten der Wissensressource in übersichtlicher Form mit einer einfachen Syntax darzustellen. Die

Sprache ist dabei kompatibel zu RDF und viele der externen Systeme zur Verarbeitung von RDF können auch mit Turtle kodierten Dateien umgehen und umgekehrt.

Um automatisiert auf das Wissen solcher Ressourcen zugreifen zu können, wurden parallel zu den Auszeichnungssprachen auch Anfragesprachen entwickelt. Diese Sprachen erlauben es, die Daten einer Wissensressource anzufragen und verschiedene Konzepte auf der Basis ihrer Verbindungen abzufragen. Zwei weit verbreitete Standards für solche Abfragesprachen sind *SPARQL* [3] und *RDQL* [2].

Wie später noch gezeigt werden soll, können die semantische Indexierung, standardisierte Ontologien und solche Abfragesprachen miteinander kombiniert werden. Dies ermöglicht Indexanfragen auf Basis des Wissens der Wissensressourcen.

4.7. EFGT-Netze, Ontologien und explizite Wissensressourcen

Bei der Betrachtung von Ontologien wurde darauf eingegangen, dass Thesauri und Ontologien trotz einiger Unterschiede viele Ähnlichkeiten aufweisen. Im Hinblick auf die in dieser Arbeit behandelte semantische Indexierung von Textdokumenten spielen die formalen Unterschiede zwischen Ontologien und Thesauri eine untergeordnete Rolle und werden somit in dieser Arbeit gleich behandelt.

Für die streng hierarchisch aufgebauten EFGT-Netze kann eine ähnliche Argumentation verwendet werden, zumal das EFGT-Netz der TopicZoom Hierarchie selbst als RDF Dokument ausgezeichnet ist[10], die ja vornehmlich zur Definition von Ontologien entwickelt wurden. EFGT-Netze können genauso wie Thesauri und Ontologien auch als Wissensressourcen angesehen werden, in denen die Fakten als einfache Subjekt-Prädikat-Objekt-Tripel abgelegt sind. Sie beinhalten im Vergleich zu Ontologien ein einziges (transitives) Prädikat *„ist allgemeiner als"* (bzw. *„ist spezieller als"*) und nicht eine Reihe von unterschiedlichen Prädikaten mit verschiedenen Eigenschaften.

Der wesentliche Unterschied zwischen den verschiedenen Formalismen liegt in den Prädikaten der jeweiligen Wissensressourcen begründet. Während das alleinige Prädikat der EFGT-Netze implizit transitiv ist und somit die transitive Hülle der gesamten Relation einmalig berechnet werden kann, können bei den verschiedenen Prädikaten einer Ontologie weitere Relationen in unterschiedlicher Weise abgeleitet werden.

[10] Auch weist die TopicZoom Hierarchie neben dem einen Prädikat R, unter dem die eigentlichen Konzepte des Netzes untereinander verbunden sind, weitere Prädikate auf, die zur Auszeichnung natürlichsprachlicher Ausdrücke für die Konzepte verwendet werden.

Konzeptuell werden die unterschiedlichen Formalismen bei der semantischen Indexierung gleich behandelt. Dies soll aber nicht heißen, dass es keine Unterschiede zwischen den hierarchischen EFGT-Netzen und verschiedenen Ontologien, wie etwa der Ontologie „Erster Weltkrieg“ gibt. Die *Eigenschaften* der Formalismen sind durchaus unterschiedlich. Es ergeben sich somit unterschiedliche Möglichkeiten und Einschränkungen bei der Abfrage des Index. Auch beeinflusst die genaue Art der Wissensressourcen die konkrete Realisation des Indexes.

Das im nächsten Kapitel vorgestellte Grundverfahren zur semantischen Indexierung mit expliziten Wissensressourcen baut auf der, in Kapitel 1.5 dargestellten, Definition 1.5.1 von Wissensressourcen auf. Diese Darstellung expliziter Wissensressourcen besitzt dabei wie EFGT-Netze und Ontologien eine Konzeptmenge sowie eine Menge binärer Relationen auf diesen. Im Gegensatz dazu verfügen weder EFGT-Netze noch Ontologien über ein Lexikon.

Sowohl das TopicZoom Netz als auch die Ontologie „Erster Weltkrieg“ verfügen aber beide über Zuordnungen, die ihren Konzepten verschiedene natürlichsprachliche Ausdrücke zuweisen. Zur Verwendung dieser beiden Ressourcen zur semantischen Indexierung mit expliziten Wissensressourcen müssen eben diese Zuordnungen heraus gelöst werden und in ein entsprechendes Lexikon mit entsprechenden Konzeptzuweisungen umgewandelt werden. Dies geschieht bei der semantischen Indexierung beim Parsen der expliziten Wissensressourcen, wo entsprechende Einträge nicht in die Relations- bzw. Konzeptmenge aufgenommen werden, sondern zum Aufbau des Lexikon und der Konzeptzuweisungen verwendet werden.

Es bleibt noch anzumerken, dass auch andere Ontologien, die nicht im Hinblick auf die hier dargestellte Form der semantischen Indexierungen erzeugt wurden, oft über verschiedene Relationen verfügen, die ihren Konzepten natürlichsprachliche Bezeichner zuordnen [28, 4, 1, 57]. Solche Prädikate können immer bei der semantischen Indexierung zum Aufbau der Lexika herangezogen werden.

4.8. Zusammenfassung

In diesem Kapitel wurde eine Reihe von unterschiedlichen Formalismen vorgestellt, mit denen Wissen repräsentiert wird. Einerseits wurde mit den EFGT-Netzen ein streng hierarchischer Formalismus dargestellt. Andererseits wurden auch flexiblere Ansätze mittels Ontologien erläutert und ebenso Formalismen zur vereinfachten Erstellung von Ontologien erwähnt. Sowohl für Ontologien als auch für EFGT-Netze wurden kleine Beispiele angegeben, die im weiteren Verlauf der Arbeit dazu dienen, verschiedene Aspekte der semantischen Indexierung zu erläutern.

In diesem Kapitel wurden unterschiedlicher Formen der Repräsentation von Wissensressourcen behandelt. Es ist aber nicht Gegenstand dieser Arbeit die Repräsentation bzw. den Aufbau von Wissensressourcen zu diskutieren. Es sollten lediglich die Unterschiede und vor allem auch die Gemeinsamkeiten verschiedener Formalismen zur Wissensrepräsentation dargelegt werden. So ist es möglich, Wissensressourcen unterschiedlicher Formalismen als explizite Wissensressourcen zur semantischen Indexierung zu verwenden. Im nächsten Kapitel wird nun das Grundverfahren der semantischen Indexierung mit expliziten Wissensressourcen dargestellt.

5. Grundverfahren zur semantischen Indexierung mit expliziten Wissensressourcen

Die semantische Indexierung erzeugt aus einer Menge von Dokumenten einen durchsuchbaren und abfragbaren Index. In diesem Kapitel wird nun die semantische Indexierung mit expliziten Wissensressourcen beschrieben. Dabei werden insbesondere die Struktur des semantischen Index, dessen Aufbau aus den Eingabedokumenten und die Anfragemöglichkeiten an den resultierenden Index beschrieben. Alle diese Aspekte der semantischen Indexierung werden dabei immer auch in Hinblick auf die explizite Wissensressourcen aus EFGT-Netzen und Ontologien dargestellt. Ebenso wird auf alternative Indexstrukturen eingegangen und es werden verschiedene praktische Aspekte der Implementierung beleuchtet.

5.1. Semantische Indexierung

Die grundlegende Annahme bei der hier betrachteten Form der semantischen Indexierung ist, dass in natürlichsprachlichen Dokumenten eine Reihe abstrakter Konzepte verborgen ist, deren Gesamtheit die thematische Einordnung von Dokumenten bildet. Diese abstrakten Konzepte können über entsprechende natürlichsprachliche Zeichenfolgen im Text identifiziert, ihren Konzepten in den expliziten Wissensressourcen zugeordnet und in den Index eingetragen werden. Hierbei bilden das Lexikon und dessen konzeptuelle Zuweisungen in der Wissensressource die Verbindung zwischen natürlichsprachlichen Textbausteinen in Dokumenten und abstrakten Konzepten der Wissensressource. Die Grundlagen der semantischen Indexierung speziell mit EFGT-Netzen (vgl. Kapitel 4.3) sowie eine ausführlichere Beschreibung verschiedener Implementierungsaspekte findet sich in [19].

Bei der semantischen Indexierung mit expliziten Wissensressourcen wird eine Dokumentenkollektion mit einer expliziten Wissensressource (semantisch) indexiert. Dabei wird der semantische Index aus einer expliziten Wissensressource $\mathcal{W}$

(vgl. Definition 1.5.1) und Dokumenten $d \in \mathcal{D}$ einer Dokumentenkollektion $\mathcal{D}$ aufgebaut, wobei die Dokumente $d \subseteq \Sigma^*$ der Dokumentenkollektion hier als einfache Ketten von Buchstaben aufgefasst werden.

Bei der semantischen Indexierung werden nicht nur die direkt in den Texten vorgefundenen Konzepte indexiert, sondern auch deren Verbindungen in den Relationen der Relationsmenge der Wissensressource. Es werden also auch Konzepte indexiert, die mit den ursprünglich vorgefundenen Konzepten in der Wissensressource verbunden sind. Diese zusätzlichen Indexeinträge bilden das semantische Wissen der Wissensressource im Index ab und erlauben effiziente, thematische Suchen auf dem Index. Die Verbindungen der Konzepte stammen dabei direkt aus den in den Wissensressourcen modellierten Beziehungen.

Für Suchanfragen auf dem Index stehen dann nicht nur die direkt in den Dokumenten gefundenen Konzepte zur Abfrage bereit sondern auch die von diesem abgeleiteten Konzepte. Wenn zwei Konzepte $A, B \in \mathcal{W}_K$ in einer Relation $R \in \mathcal{W}_R$ in einer Wissensressource $\mathcal{W} = \langle \mathcal{W}_K, \mathcal{W}_R, \mathcal{W}_{Lex}, \kappa \rangle$ miteinander verbunden sind — es gilt somit $R(A, B)$ — und wenn das Konzept A in einem Dokument $d \in \mathcal{D}$ der Dokumentenkollektion gefunden wird, dann wird nicht nur das Vorkommen von A in d indexiert sondern auch ein Vorkommen von B in d. Das Vorkommen von A in d heißt dann *direkter Treffer* bzw. *direktes Vorkommen von A in d*. Dem gegenüber wird das Vorkommen von B in d als *indirekter Treffer* bzw. als *indirektes Vorkommen von B in d* bezeichnet. A wird dann auch als *Ursprungskonzept von B*, und B als *abgeleitetes Konzept von A* bezeichnet.

Bei der semantischen Indexierung werden indirekte Treffer für ein Konzept B nur dann erfasst, wenn B mit dem direkt gefundenen Konzept A explizit über eine kodierte Relation verbunden ist. Ableitbare Beziehungen werden demnach nicht berücksichtigt.

Bei einer späteren Anfrage an den Index nach einem Konzept B können so auch Vorkommen von A mit als mögliche Suchtreffer berücksichtigt werden. Die genauen Folgen für die Suche bei einem solchen Vorgehen liegen dabei vor allem in den tatsächlich in den Wissensressourcen modellierten logisch-semantischen Verbindungen der Relationen und Konzepte.

Beispiel 5.1.1. Eine Wissensressource enthalte unter anderem die beiden Konzepte *Ludwig van Beethoven* und *Johann Sebastian Bach*, die beide in einer Relation der Wissensressource mit dem Konzept *Komponist* verbunden sind. Bei der Indexierung werden nun nicht nur die Vorkommen von *Ludwig van Beethoven* und *Johann Sebastian Bach* in den Dokumenten indexiert, sondern auch immer ein indirektes Vorkommen des Konzepts *Komponist*.

Bei einer Suchanfrage nach dem Konzept *Komponist* können nun auch Dokumente bzw. Textstellen zurückgeliefert werden, in denen das Konzept *Komponist*

selbst nicht direkt aufgetaucht ist, sondern nur das Konzept *Johann Sebastian Bach* oder *Ludwig van Beethoven*.

Das in der Wissensressource modellierte Wissen, nämlich dass *Johann Sebastian Bach und Ludwig van Beethoven Komponisten sind*, ist durch die Verbindungen der Wissensressource explizit mit in den Index eingeflossen und steht daher bei Suchanfragen direkt zur Verfügung.

Im Allgemeinen wird bei der semantischen Indexierung davon ausgegangen, dass bei den Relationen der Wissensressource *semantisch engere* Konzepte auf *semantische weitere* Konzepte verweisen. Gilt $R(A, B)$, so wird angenommen, dass Konzept B allgemeiner ist als Konzept A. Ein direkter Treffer für A führt zu einem indirekten Treffer für B, aber nicht umgekehrt. Durch das Hinzufügen der verbundenen Konzepte in den semantischen Index können so Suchanfragen nach Oberbegriffen Vorkommen zu Unterbegriffen liefern. So liefert eine Suchanfrage nach dem allgemeineren Konzept B auch Textbelege zu Vorkommen des Konzepts A.

Diese Forderung an die Verbindungen der Wissensressource ist nicht bindend und auch keine Grundvoraussetzung der semantischen Indexierung. Es ist durchaus möglich Verbindungen, die nicht diesem strikten Schema folgen, semantisch zu indexieren. In solchen Fällen müssen allerdings die Auswirkungen dieser Verbindungen vor allem in Hinblick auf die spätere Suche auf dem Index untersucht und gegebenenfalls angepasst werden.

Anders als bei EFGT-Netzen, die eine strikte hierarchische Struktur von speziellen Themen hin zu allgemeinen Themen abbilden, ist diese allgemeine Voraussetzung im Falle von Ontologien, die ja im Allgemeinen weniger stark formalisiert sind als EFGT-Netze, nicht immer gegeben. Bei Ontologien müssen die verschiedenen Verbindungen auf das allgemeine Vorgehen und die Auswirkungen auf die semantische Indexierung hin untersucht werden. Gegebenenfalls müssen bestimmte Verbindungen der Ontologie angepasst oder sogar ignoriert werden.

Ontologien stellen das Wissen einerseits über die Verbindungen der Konzepte untereinander und andererseits über die spezifischen Relation der Verbindung dar. Um das über die Relationen abgebildete Wissen ebenfalls in den semantischen Index aufnehmen zu können, werden bei der semantischen Indexierung mit Ontologien neben den Verbindungen von Konzepten auch die vorliegende Relation einer Verbindung indexiert, um auf die semantische Information der Relationen auch bei Suchanfragen zugreifen zu können.

Beispiel 5.1.2. Ein Konzept A werde in einem Dokument d gefunden. Das Konzept A sei Teil der in Abbildung 5.1b dargestellten Ontologie. Es wird bei der semantischen Indexierung zuerst das direkte Konzept A indexiert. Zusätzlich werden dann zwei weitere, indirekte Konzepte indexiert. Diese indirekten Konzepte sind

diejenigen, die mit dem Ursprungskonzept A im Graphen der Ontologie verbunden sind. Bei der semantischen Indexierung wird somit das direkte Konzept A und die beiden indirekten Konzepte B unter der Relation a und C unter der Relation b indexiert.

5.2. Indexierungsressourcen

Für die semantische Indexierung werden eine Wissensressource $\mathcal{W} = \langle \mathcal{W}_K, \mathcal{W}_R, \mathcal{W}_{Lex}, \kappa \rangle$ im Sinne von Definition 1.5.1 und ein entsprechender $\mathcal{W}$-Automat $= \langle \Sigma, Q, s, \delta, F, \theta, t \rangle$ im Sinne von Definition 1.5.3 verwendet.

Ontologien und EFGT-Netze unterscheiden sich in dieser Darstellung hauptsächlich in Bezug auf ihre Relationsmengen $\mathcal{W}_R$ der Wissensressource. EFGT-Netze verfügen über eine Relationsmenge $\mathcal{W}_R = \{R\}$, die aus einer einzelnen Relation R besteht. Diese Relation R ist dabei transitiv und antisymmetrisch. Dagegen verfügen Ontologien meist über mehrere Relationen $\mathcal{W}_R = \{R_1, \ldots, R_n\}$, die allesamt unterschiedliche, nicht näher spezifizierte Eigenschaften besitzen.

Die Relationen von Ontologien können beliebige Eigenschaften und Verbindungen aufweisen. Da aber Verbindungen von Konzepten mit sich selbst in Hinblick auf die semantische Indexierung wenig sinnvoll sind und den semantischen Index unnötig aufblähen[1], können solche Verbindungen schon beim Aufbau der Wissensressource aus den Relationen entfernt werden.

Alle Konzepte $k \in \mathcal{W}_K$ der Wissensressource verfügen über eine eindeutige interne Identifikationsnummer $k_{id} > 0$. Die Relationen $R \in \mathcal{W}_R$ der Wissensressource verfügen ebenfalls über eindeutige Identifikationsnummern $r_{id} > 0$. Diese Identifikationsnummern werden beim Aufbau der Wissensressource jeweils für die Konzepte und Relationen erzeugt und dienen der semantischen Indexierung intern zur eindeutigen Identifizierung von Konzepten und Relationen. Neben diesen Identifikationsnummern verfügen sowohl die Konzepte als auch die Relationen über eindeutige, externe URIs, die sie auch nach außen eindeutig identifizieren.

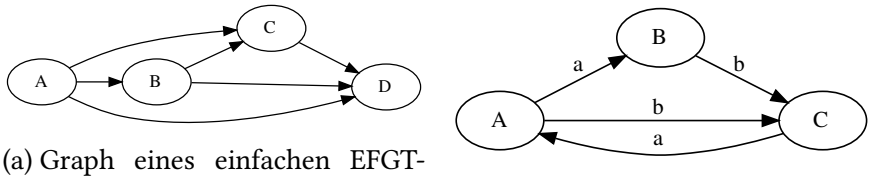
5.2.1. Relationen der Wissensressource

Intern sind die Relationen $R \in \mathcal{W}_R$ der Wissensressource als *gerichtete Graphenstruktur* realisiert. Diese Struktur besteht aus einer Menge von *Knoten*[2], die untereinander wiederum über eine Menge von *Kanten*[3] miteinander verbunden sind. Die

[1] Für jeden Eintrag zu einem gefundenen Konzept müsste in diesem Fall der gleiche indirekte Eintrag für das gleiche Konzept erzeugt werden.

[2] Diese repräsentieren die Konzepte der expliziten Wissensressource.

[3] Diese repräsentieren die Relationen der expliziten Wissensressource.



(a) Graph eines einfachen EFGT-Netzes

(b) Graph einer einfachen Ontologie

Abbildung 5.1.: Repräsentation der Relationen von Wissensressourcen als gerichteter Graph.

Kanten des Graphen sind gerichtet und können optional ein Label tragen, welches das Prädikat der Verbindung zwischen den beiden Knoten angibt. Dieses Label spezifiziert die Relation, zu der eine Verbindung zwischen zwei Knoten gehört.

Sofern die Wissensressource nur über eine Art von Verbindung verfügt oder falls die genaue Art der Verbindung in der Wissensressource keine Rolle spielt, sind die Label der Kante nicht ausgezeichnet und werden bei der semantischen Indexierung ignoriert. Es wird in diesem Fall dann einfach angenommen, dass die Relation unter der irgendwelche Verbindungen auftauchen, keine semantische Information enthält und alle Verbindungen aus derselben Relation stammen.

Alle Knoten des Graphen sind immer eindeutig mit einem Konzept $k \in \mathcal{W}_K$ der Wissensressource identifiziert. Jeder Knoten des Graphen entspricht genau einem Konzept der Wissensressource und umgekehrt. Hierzu werden die Identifikationsnummern k_{id} der Konzepte der Wissensressource verwendet. Jeder Knoten des Graphen trägt eine Identifikationsnummer, die immer genau der Identifikationsnummer des zugeordneten Konzepts entspricht.

Die Kanten des Graphen repräsentieren die Verbindungen zwischen den Konzepten der Relationen der Wissensressource $\mathcal{W}_R$. Im Falle von EFGT-Netzen stammen alle Verbindungen aus einer einzigen Relation und die Verbindungen tragen kein Label. Bei Ontologien sind die Relationen, aus denen die Verbindungen im Graphen stammen, explizit als Label an den Kanten des Graphen ausgezeichnet.

Beispiel 5.2.1. Abbildung 5.1a zeigt den Graphen eines einfachen EFGT-Netzes mit den Verbindungen seiner transitiven, antisymmetrischen Relation $R = \{\langle A, B\rangle, \langle A, C\rangle, \langle A, D\rangle, \langle B, C\rangle, \langle B, D\rangle, \langle C, D\rangle\}$.

Abbildung 5.1b zeigt den Graphen einer Ontologie mit der Relationsmenge $\mathcal{W}_R = \{R_a, R_b\}$ und den Relationen $R_a = \{\langle A, B\rangle, \langle C, A\rangle\}$ und $R_b = \{\langle B, C\rangle, \langle A, C\rangle\}$. Im Gegensatz zum Graphen des EFGT-Netzes tragen die Kanten ein Label, welches die Zugehörigkeit einer Verbindung zu einer bestimmten Relation $R \in \mathcal{W}_R$ der Wissensressource angibt.

5.2.2. Das Lexikon der Wissensressource

Das Lexikon der Wissensressource $\mathcal{W}_{Lex}$ ist als $\mathcal{W}$-Automat $= \langle \Sigma, Q, s, \delta, F, \theta, t \rangle$ im Sinne von Definition 1.5.3 realisiert. Der Automat selbst ist minimiert (vgl. Kapitel 3.4.2). Die Konzeptzuweisungen κ und die Typisierungsfunktion t sind über die eindeutigen Identifikationsnummern der Konzepte k_{id} implementiert und sind direkt in den finalen Zustandszellen des Automaten gespeichert.

Das Lexikon der Wissensressource umfasst nicht nur die unterschiedlichen Vollformen einzelner Wörter, sondern kann auch beliebig lange, komplexe Mehrwortverbindungen enthalten. Je nach verwendeter Wissensressource können die Lexikoneinträge auch Einträge zu unterschiedlichen natürlichen Sprachen umfassen.

Bei der Suche von Konzepten auf natürlichsprachlichen Texten werden die Texte *normalisiert* (vgl. Kapitel 5.4.1). Um sicher zu stellen, dass die Lexikoneinträge mit den normalisierten Texten zusammenpassen und auch wirklich in den Texten gefunden werden können, müssen die Lexikoneinträge auf dieselbe Weise normalisiert werden wie die Texte.

Im Allgemeinen zeigen mehrere Lexikoneinträge auf dasselbe Konzept. Es gibt aber durchaus auch Konzepte, die nur von einem oder auch von keinem Lexikoneintrag referenziert werden. Knoten ohne Lexikoneintrag können bei der Indexierung niemals direkt in den Dokumenten gefunden werden. Sie können aber bei der Indexerzeugung durch die Verbindungen im Graphen als indirekte Konzepte (vgl. Kapitel 5.3.1) indexiert werden. Bei Suchanfragen (vgl. Kapitel 5.5) können sie immer über ihre eindeutige URI referenziert werden.

Beispiel 5.2.2. Ein Beispiel für einen solchen Knoten ist der, in Kapitel 4.3.2 erwähnte, *Topnode* der TopicZoom-Hierarchie. Dieser Knoten verfügt über keinen eigenen Lexikoneintrag und kann daher niemals direkt in den Dokumenten gefunden werden.

Aufgrund der Transitivität der Relation wird er allerdings von jedem Knoten im Graphen direkt referenziert und jedes gefundene Konzept erzeugt immer auch ein indirekten Eintrag für dieses Konzept.

5.3. Struktur des Index

Der semantische Index besteht aus einer Menge sogenannter *Postingfiles*, welche alle immer eindeutig einem Konzept $k \in \mathcal{W}_K$ der Wissensressource zugeordnet sind. Jedes Postingfile enthält dabei nur *direkte* und *indirekte* Einträge zu einem einzigen Konzept. Jeder Eintrag eines Postingfiles referenziert dabei eine eindeutige Textstelle — einen sogenannten *Textbeleg* — in einem Dokument der Dokumentenkollektion. Die Gesamtheit aller Postingfiles bildet dann den semantischen

Index.

Die Dokumentenkollektion bezeichnet die Menge der Dokumente, die bei der semantischen Indexierung indexiert werden. Jedes Dokument dieser Kollektion verfügt über einen eindeutigen Pfad, der es eindeutig identifiziert.

5.3.1. Direkte und indirekte Indexeinträge

Direkte Indexeinträge stellen Textbelege zu direkten Vorkommen von Konzepten dar, deren Lexikoneinträge direkt in einem Dokument der Kollektion gefunden wurden. Sie werden immer in das Postingfile des Konzepts geschrieben, dessen Lexikoneintrag in einem Dokument gefunden wurde.

Indirekte Indexeinträge werden dagegen in den Postingfiles der abgeleiteten Konzepte eines Ursprungskonzepts geschrieben, mit denen das Ursprungskonzept, durch das ein direkter Treffer resultierte, im Graphen verbunden ist. Für jedes Konzept, mit dem das Ursprungskonzept im Graphen der Wissensressource verbunden ist, wird der gleiche Textbeleg in das Postingfile des abgeleiteten Konzepts als indirekter Indexeintrag geschrieben.

Direkte und indirekte Indexeinträge eines Ursprungskonzepts verweisen immer auf denselben Textbeleg, stehen aber in unterschiedlichen Postingfiles der direkten respektive indirekten Konzepte. Jeder Indexeintrag eines Postingfiles kann immer eindeutig als entweder direkt oder indirekt identifiziert werden.

Bei indirekten Indexeinträgen von EFGT-Netzen muss die Ursprungsrelation der Verbindung nicht explizit im Index abgespeichert werden, da diese ja implizit bekannt ist. Im Falle von Ontologien wird, wie bereits erwähnt, auch die Relation der Verbindung im Index abgelegt.

Indexeinträge von EFGT-Netzen enthalten eine spezielle Markierung, welche angibt ob ein Eintrag direkt oder indirekt ist, Indexeinträge von Ontologien dagegen enthalten jeweils die Identifikationsnummer der Ursprungsrelation der Verbindung, die im Falle von direkten Einträgen 0 ist und im Falle von indirekten Einträgen der Identifikationsnummer der entsprechenden Relation $r_{id} > 0$ entspricht.

Diese *Vererbung von Indexeinträgen*, die zur Erzeugung von direkten und indirekten Indexeinträgen führt, stellt die primäre Technik dar, um das in einer Wissensressource repräsentierte Wissen im Index zu speichern und für die Suche bereitzustellen. Sowohl die direkten als auch die indirekten Indexeinträge stellen einen zentralen Bestandteil der semantischen Indexierung und des semantischen Index dar.

Beispiel 5.3.1. In einem Dokument $d \in \mathcal{D}$ der Dokumentenkollektion wird der String *„Erich von Falkenhayn“* gefunden, der dem entsprechenden Konzept *Erich von Falkenhayn* zugeordnet ist (vgl. Abbildung 4.4).

Es wird nun ein Indexeintrag E erzeugt, der eindeutig auf das Vorkommen des Strings *„Erich von Falkenhayn“* in dem Dokument d verweist. Dieser Eintrag E wird nun einerseits als direkter Indexeintrag in das Postingfile des Konzepts *Erich von Falkenhayn* geschrieben. Die Identifikationsnummer der Ursprungsrelation des direkten Indexeintrags beträgt 0.

Genauso wird derselbe Eintrag E nun auch als indirekter Eintrag mit seinen entsprechenden Ursprungsrelationen in die Postingfiles der verbundenen Konzepte geschrieben: *DEU* aus der Relation *Country*, *1861* aus der Relation *Birth*, *1922* aus der Relation *Death*, *General* aus der Relation *Occupation* und *Military* und *Politics* aus der Relation *Scope*. Jeder indirekte Indexeintrag wird in die Postingfiles der entsprechenden indirekten Konzepte geschrieben und enthält immer eine Identifikationsnummer $r_{id} > 0$ der entsprechenden Ursprungsrelationen.

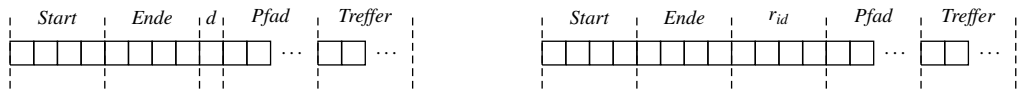
5.3.2. Format der Indexeinträge

Jeder Indexeintrag eines Postingfiles referenziert immer eine eindeutige Textstelle in einem Dokument der Dokumentenkollektion. Alle Informationen, die in einem Indexeintrag abgespeichert sind, stehen bei der späteren Abfrage des Index immer direkt ohne weitere Berechnungen zur Verfügung.

Hierzu besteht jeder Eintrag aus der im Dokument gefundenen Zeichenfolge und zwei Textpositionen, die jeweils die Start- und Endpositionen des gefundenen Lexikoneintrags im Text angeben. Des Weiteren enthält jeder Eintrag den eindeutigen Pfad des Dokuments, aus dem er stammt.

Um zwischen direkten und indirekten Indexeinträgen von EFGT-Netzen unterscheiden zu können, wird zusätzlich in einem weiteren Feld abgelegt ob es sich bei dem Eintrag um einen direkten oder indirekten Indexeintrag handelt (Abbildung 5.2a). Soll bei Ontologien die Ursprungsrelation eines Treffers mit indexiert werden, wird dieses Feld dazu verwendet, die Identifikationsnummer der Relation r_{id} zu speichern (Abbildung 5.2b). Direkte Indexeinträge sind dann mit einer 0 markiert.

Jeder Indexeintrag enthält die Start- und Endposition des Suchtreffers im Dokument. Mit diesen Informationen und mit dem Pfad des Dokuments können die Suchtreffer und auch deren Kontext aus den Indexeinträgen rekonstruiert werden. Hierzu muss allerdings das Dokument, welches der Indexeintrag referenziert, neu eingelesen werden. Die zusätzliche Speicherung des Suchtreffers im Index ermöglicht es, den Suchtreffer eines Indexeintrags schnell darzustellen, ohne das Ursprungsdokument erneut einlesen zu müssen.



(a) Indexeintrag für EFGT-Netze (b) Indexeintrag für Ontologien

Abbildung 5.2.: Allgemeiner Aufbau von Indexeinträgen. Jedes Kästchen entspricht dabei einem Byte in der Indexdatei. Die Felder der Pfad- und Trefferangaben haben dabei jeweils eine variable Breite von mindestens 2 Bytes.

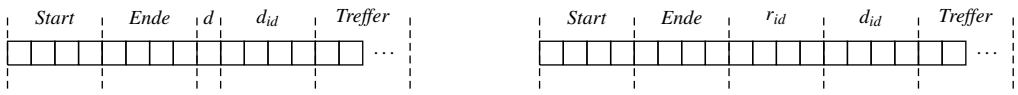
Je nach Anwendung werden diese Informationen zur Textposition und zum Suchtreffer nicht zwingend benötigt. Um den Platzbedarf des resultierenden semantischen Index zu verringern, können diese Informationen beim Schreiben der Indexeinträge weggelassen werden. Das Weglassen dieser optionalen Informationen ermöglicht es, die Größe der Postingfiles und damit die Größe des semantischen Index zu beeinflussen.

Fehlen die Textpositionen in Indexeinträgen, kann der Kontext von Suchtreffern nicht mehr einfach rekonstruiert werden. Fehlen die Suchtreffer in den Indexeinträgen, kann der Suchtreffer nur rekonstruiert werden, indem das Dokument neu eingelesen wird und der Suchtreffer über die Textpositionen im Dokument rekonstruiert wird. Fehlen beide Informationen kann weder der Kontext noch der ursprüngliche Suchtreffer im Dokument rekonstruiert werden. Es ist nur noch möglich, zwischen direkten und indirekten Treffern in den unterschiedlichen Dokumenten zu unterscheiden.

Die vorliegende Implementierung der semantischen Indexierung erlaubt es, die optionalen Informationen nicht mit in den Index zu schreiben. Es können beide optionale Informationen nur die Suchtreffer, nur die Textpositionen oder keine der beiden Informationen in den Index aufgenommen werden. So kann zwischen der im Index verfügbaren Information und dessen Größe abgewogen werden.

Je nach verwendeter Wissensbasis können die Ursprungsrelationen von indirekten Indexeinträgen gespeichert werden oder nicht. Im Falle von EFGT-Netzen ist die Ursprungsrelation implizit immer bekannt und muss nicht zusätzlich mit in den Indexeinträgen gespeichert werden. Bei der Verwendung von Ontologien dagegen sollte die Ursprungsrelation mit indexiert werden, um so auch die erweiterten Anfragemöglichkeiten mit Ontologien verwenden zu können (vgl. Kapitel 5.5.3).

Eine letzte Optimierung hinsichtlich der Größe des resultierenden Index ist, die Dokumentenpfade nicht mehr in jedem einzelnen Indexeintrag zu speichern, sondern diese in einem Dokumentenregister zu verwalten. Das Dokumentenregister verwaltet die Pfade der Dokumente und ordnet jedem Dokument eine eindeutige



(a) Indexeintrag für EFGT-Netze mit Dokumentenidentifikationsnummer

(b) Indexeintrag für Ontologien mit Dokumentenidentifikationsnummer

Abbildung 5.3.: Allgemeiner Aufbau von Indexeinträgen mit Dokumentenidentifikationsnummern. Jedes Kästchen entspricht dabei einem Byte in der Indexdatei. Das Feld der Treffer hat dabei jeweils eine variable Breite von mindestens 2 Bytes.

Dokumentenidentifikationsnummer d_{id} zu.

Auf diese Weise müssen die Pfade nicht mehr in den Indexeinträgen mit gespeichert werden. Anstatt des Dokumentenpfades enthält nun jeder Indexeintrag ein Feld fester Breite zur Speicherung der entsprechenden Dokumentenidentifikationsnummer (vgl. Abbildung 5.3).

Die Verwaltung des Dokumentenregisters verkompliziert den Index, da dieser nun neben den Postingfiles auch das Dokumentenregister verwalten muss und eindeutige Zuordnungen zwischen Dokumentenpfaden und Dokumentenidentifikationsnummern gewährleisten muss. Gerade aber wenn die Dokumentenpfade sehr lang sind oder wenn auch Webseiten mit ihren URIs indexiert werden, kann durch die Verwendung eines Dokumentenregisters die Größe des resultierenden Index stark reduziert werden.

Für alle weiteren Betrachtungen spielt es keine Rolle, wie der Dokumentenpfad in den Indexeinträgen abgespeichert wird. Die Dokumentenpfade können immer entweder direkt über den abgespeicherten Pfad oder indirekt über das Dokumentenregister des Index und der Dokumentenidentifikationsnummer d_{id} des Indexeintrags ausgelesen werden.

5.3.3. Postingfiles

Die Postingfiles stellen die zentrale Datenstruktur dar, aus der der semantische Index aufgebaut ist. Jedes Postingfile enthält eine Menge von direkten und indirekten Indexeinträgen, die alle immer zum gleichen Konzept gehören.

Die Indexeinträge eines Postingfiles referenzieren unterschiedliche Textstellen in unterschiedlichen Dokumenten der Kollektion. Dabei können durchaus auch mehrere Indexeinträge auf verschiedene Textstellen eines Dokuments verweisen[4].

[4]Abhängig von der Wissensressource ist es auch möglich, dass ein Postingfile einen direkten und einen indirekten Eintrag, die beide auf die gleiche Textposition im selben Dokument verwei-

Um alle Textstellen in allen Dokumenten der Dokumentenkollektion, die einem bestimmten Konzept zugeordnet sind, einzulesen, muss das dem Konzept zugehörige Postingfile identifiziert werden und dann alle Indexeinträge ausgelesen werden.

Alle Postingfiles liegen unterhalb einer, als *Indexverzeichnis* bezeichneten, Verzeichnisstruktur. Sie können entweder durch die internen Identifikationsnummer der Konzepte, die auch für die konzeptuellen Zuordnungen der Lexikoneinträge verwendet werden, oder durch deren URIs identifiziert werden.

Die Postingfiles können bei der Indexerzeugung iterativ aus den Dokumenten aufgebaut werden und es können auch weitere Dokumente im Nachhinein in den Index eingefügt werden. Es müssen — anders als bei der klassischen Indexerzeugung — nicht erst die Schlagwörter der verschiedenen Dokumente gesammelt werden und dann in eine invertierte Liste überführt werden, aus der dann der eigentliche Index aufgebaut wird [66, S.109ff].

Sobald ein Lexikoneintrag in einem Dokument identifiziert wurde, kann das Verfahren einen Indexeintrag erzeugen und diesen sogleich in die entsprechenden Postingfiles schreiben. Für jeden Lexikoneintrag in einem Dokument sind über die konzeptuellen Zuordnungen des Lexikons der Wissensressource, die Knoten des Graphen und somit auch all seine Verbindungen bekannt. Der entsprechende direkte Eintrag kann sofort in das entsprechende Postingfile geschrieben werden. Über die Kanten des Knotens können sogleich auch alle indirekten Einträge in die entsprechenden Postingfiles geschrieben werden.

Beispiel 5.3.2. Bei der semantischen Indexierung eines Dokuments mit dem in Abbildung 5.1a dargestellten EFGT-Netz wird über einen entsprechenden Lexikoneintrag das Konzept *A* im Dokument identifiziert. Es wird nun sofort ein direkter Indexeintrag in das Postingfile von Konzept *A* geschrieben und weitere indirekte Indexeinträge in die Postingfiles der Konzepte *B*, *C* und *D*.

Als nächstes wird das Konzept *B* im Dokument identifiziert, ein direkter Indexeintrag in das Postingfile des Konzepts *B* geschrieben und jeweils ein indirekter Eintrag in die Postingfiles der Konzepte *C* und *D*.

Es werden keine weiteren Konzepte in dem Dokument gefunden und die Indexierung des Dokuments ist beendet. Der Index wurde bereits während der Indexierung des Dokuments aufgebaut und alle Einträge sind bereits in die entsprechenden Postingfiles geschrieben worden. Es kann sofort mit der Indexierung der Kollektion fortgefahren werden und das nächste Dokument der Dokumentenkollektion indexiert werden oder auch Suchanfragen an den Index abgearbeitet werden.

sen, enthält. Dies passiert genau dann, wenn ein Konzept der Wissensressource mit sich selbst verbunden ist.

5.3.4. Semantischer Index

Der semantische Index besteht aus der Menge der Postingfiles zu den verschiedenen Konzepten der Wissensressource. Alle Postingfiles des Index liegen unterhalb einer Verzeichnisstruktur. Jedes Konzept, das während der Indexierung direkt über einen Lexikoneintrag in einem Dokument identifiziert wurde und jedes Konzept, welches von solch einem Konzept referenziert wird, verfügt über ein eigenes Postingfile im semantischen Index.

Konzepte ohne ein entsprechendes Postingfile im semantischen Index konnten weder direkt in den Dokumenten der Kollektion identifiziert werden, noch wurden sie von irgendwelchen anderen, in den Dokumenten gefundenen Konzepten referenziert. Zu diesen Konzepten existieren weder direkte noch indirekte Textbelege in der Dokumentenkollektion.

Sinnvolle Anfragen an den semantischen Index können nur auf Basis einzelner Konzepte der Wissensressource geschehen. Für ein angefragtes Konzept muss dass entsprechende Postingfile des semantischen Index identifiziert werden und die Indexeinträge des Postingfiles ausgelesen werden.

5.4. Indexerzeugung

Die semantische Indexierung erzeugt aus einer *Dokumentenkollektion* $\mathcal{D}$ einen semantischen Index. Die Dokumentenkollektion ist dabei einfach eine Menge von Dokumenten. Jedes Dokument $d \in \mathcal{D}$ der Dokumentenkollektion kann eindeutig über einen *Pfad* identifiziert werden. Die Pfade der Dokumente sind entweder eindeutige Pfade in einem lokalen Dateisystem oder eindeutige, globale URIs, die auf eine global verfügbare Ressource (siehe Kapitel 4.2) referenzieren.

Bei den indexierten Dokumenten der Dokumentenkollektion kann es sich einerseits um Dateien in einem Dateisystem handeln. Andererseits können aber auch Webseiten und andere durch URIs repräsentierte Ressourcen indexiert werden. Diese Dokumente müssen nicht physikalisch auf dem entsprechenden Rechner vorliegen, sondern können direkt über ihre URI angesprochen werden und so in den Index aufgenommen werden. Die Dokumente müssen einzig eindeutig durch eine URI repräsentiert sein und lokal auf dem Rechner oder global verfügbar sein.

Jedes Dokument muss bei der Indexierung in einem ersten Schritt nach Lexikoneinträgen im Lexikon der Wissensressource durchsucht werden (siehe unten). Die vorgefundenen Lexikoneinträge werden zuerst ihren Konzepten der Wissensressource zugeordnet, entsprechende direkte und indirekte Indexeinträge erzeugt und dann in die entsprechenden Postingfiles der direkten und indirekten Konzepte geschrieben.

Der Index kann iterativ aufgebaut werden. Da die semantische Indexierung den Index direkt aus den Lexikoneinträgen in den Dokumenten aufbauen kann, können nach und nach Dokumente in den Index eingefügt werden. Es können zu jedem beliebigen späteren Zeitpunkt weitere Dokumente in den Index eingefügt werden, ohne den gesamten Index neu berechnen zu müssen.

Genau wie ein bestehender Index mit weiteren Dokumenten einfach erweitert werden kann, kann eine bestehende Wissensressource um weitere Konzepte, Relationen, Verbindungen und Lexikoneinträge erweitert werden. Aus offensichtlichen Gründen stehen die neuen Daten einer erweiterten Wissensressource erst für nachträglich indexierte Dokumente zur Verfügung. Um alle Dokumente des bestehenden semantischen Index mit den neuen Daten einer Wissensressource zu versehen, muss der Index neu aus der Dokumentenkollektion aufgebaut werden.

5.4.1. Suche von Lexikoneinträgen auf den Dokumenten

Der erste Schritt bei der Erzeugung des semantischen Index aus den einzelnen Dokumenten der Kollektion ist die Suche von lexikalischen Einheiten $w \in \mathcal{W}_{Lex}$ auf den Dokumenten. Dies kann als Instanz einer *multiplen Muster- bzw. Patternsuche* auf einem Dokument angesehen werden [41, S.41ff], wobei beachtet werden muss, dass sowohl die Menge der zu suchenden Muster sehr groß sein kann und dass auch die Länge einzelner Suchmuster unter Umständen sehr groß ist.

Die semantische Indexierung geht davon aus, dass Textstellen in Dokumenten über einen Lexikoneintrag immer genau einem abstrakten Konzept zugeordnet werden können. Bei der Suche von Lexikoneinträgen auf den Dokumenten werden aus diesem Grund — und auch um die Schaffung von weiteren Ambiguitäten zu vermeiden — sich überlappende Suchtreffer nicht zugelassen. Somit kann sobald die Suche einen Lexikoneintrag identifizieren konnte, mit der neuen Suche *direkt nach dem Ende des aktuellen Suchtreffers* begonnen werden. Die Suche nach den Lexikoneinträgen folgt dabei der üblichen Leserichtung in Dokumenten und — da Lexikoneinträge auch komplexe Mehrwortverbindungen umfassen können — wird versucht, die *längst-möglichen Lexikoneinträge* in den Texten zu finden. Es wird somit eine sogenannte *leftmost-longsest* Suchstrategie angewendet, bei der nach den längsten Lexikoneinträgen, die am weitesten links im Text beginnen, gesucht wird.

Ziel der multiplen Mustersuche ist es, gegeben eine Menge von Suchmustern $P = \{p_1, p_2, \ldots, p_n\}; p_1, p_2, \ldots, p_n \in \Sigma^*$, auf einem Text $T = t_1, t_2, \ldots, t_m \in \Sigma^*$ alle sich nicht überlappenden Suchmuster $p \in P$ zu identifizieren die Präfix, Suffix oder Infix des Textes T eines Dokuments $d \in \mathcal{D}$ sind. Eine formalere Definition der leftmost-longest Suchstrategie sowie der Überlappung von Suchmustern, findet sich unter anderem in [39].

Das Lexikon $\mathcal{W}_{Lex}$ der Wissensressource enthält die natürlichsprachlichen Ausdrücke der Konzepte der Wissensressource. Es kann als eine Menge von Suchmustern angesehen werden. Bei der hier benötigten Form der multiplen Mustersuche müssen nicht nur die einzelnen Lexikoneinträge bzw. Suchmuster in einem Text identifiziert werden, sondern es muss auch möglich sein, die Position des Suchtreffers und das Suchmuster des Treffers im Text zu rekonstruieren, da diese Informationen für die Erzeugung der Indexeinträge und zur konzeptuellen Zuordnung benötigt werden.

Ein Algorithmus zur multiplen Mustersuche ist die Aho-Corasick-Suche [5]. Dieser Algorithmus basiert auf einem Trie der Suchmuster und ermöglicht es, alle Suchmuster in einem Text zu identifizieren und dabei den Text nur ein einziges Mal von vorne nach hinten durchlaufen zu müssen. Es ist auch möglich, diesen Algorithmus auf einem durch verschränkte Übergangstabellen abgespeicherten Trie (vgl. Kapitel 3.3.2) anzuwenden, indem die entsprechenden *Supply-Zustände* des Algorithmus mit in die Tabellen aufgenommen und berechnet werden. So können auch große Mengen von beliebig langen Einträgen in dem Automaten effizient abgespeichert und abgefragt werden.

Der Aho-Corasick-Algorithmus lässt sich allerdings aus verschiedenen Gründen nur schwer auf die *minimierten* Automaten mit konzeptuellen Zuordnungen anwenden (vgl. Kapitel 3.4.2). Bei der multiplen Mustersuche auf den Dokumenten der Dokumentenkollektion wird daher ein vereinfachtes Verfahren zur multiplen Mustersuche angewandt, das die Tatsache ausnutzt, dass die Suche bei der semantischen Indexierung auf natürlichsprachlichen Texten abläuft und nicht auf einer willkürlichen Zeichenfolge.

Textnormalisierung

Bevor ein Dokument nach den Lexikoneinträgen des Lexikons durchsucht wird, wird der Text des Dokuments *normalisiert*. Er wird in eine Form gebracht, welche die Suche nach den Lexikoneinträgen und vor allem auch nach den komplexen Mehrwortverbindungen in dem Lexikon ermöglicht. Je nach Konfiguration werden Großbuchstaben der Texte in Kleinbuchstaben umgewandelt und verschiedene Punktuationszeichen aus dem Text entfernt.

Die hier dargestellte Normalisierung wandelt alle Großbuchstaben in Kleinbuchstaben um und entfernt alle Punktuationszeichen. Die Implementierung der semantischen Indexierung enthält einen konfigurierbaren Normalisierer, über den die genaue Form der Normalisierung eingestellt werden kann.

Wie bereits in den Kapiteln 2.3.1 und 2.3.2 dargestellt, ist es durchaus möglich, die Dokumente durch Stemming oder Lemmatisierung noch weiter vorzuverarbeiten. Das implementierte Verfahren wendet allerdings weder Stemming noch

Lemmatisierung an. Sollte ein Verfahren zur semantischen Indexierung solche Vorverarbeitungsschritte anwenden, müssen diese vor oder nach der Textnormalisierung — noch vor der eigentlichen Suche — angewendet werden. Dabei sollten die verschiedenen Implikationen auf das verwendete Lexikon und auf die möglichen Ambiguitäten in den Wissensressourcen beachtet werden.

Bei der Textnormalisierung wird ein ausgezeichnetes *Trennsymbol* $␣ \in \Sigma$ des Alphabets verwendet. Dieses Zeichen bezeichnet Start-, End- und Trennpositionen von Tokenfolgen und dient bei der späteren Suche dazu, legale Start- und Endpositionen für gefundene Folgen von Muster zu identifizieren. Dieses Zeichen ist Teil des Alphabets und es dient auch im Lexikon dazu, die Trennpositionen der Einzelwörter in Mehrwortverbindungen zu kennzeichnen. Die Hauptaufgabe der Normalisierung besteht darin, jede Folge von zwei oder mehr Trennsymbolen auf ein eindeutiges Trennsymbol abzubilden.

Ziel der Textnormalisierung ist es, den Eingabetext $T = t_1, t_2, \ldots, t_n$ so zu normalisieren, dass der normalisierte Text $N = n_1, n_2, \ldots, n_m$ folgende Bedingungen erfüllt:

1. Das erste Zeichen von N ist ein Trennsymbol.
2. Das letzte Zeichen von N ist ein Trennsymbol.
3. N enthält nur Kleinbuchstaben. Großbuchstaben in T müssen bei der Normalisierung in ihre entsprechenden Kleinbuchstaben umgewandelt werden.
4. N enthält keine Punktuationszeichen. Alle Punktuationszeichen des Textes werden durch das Trennsymbol ersetzt.
5. Auf ein Trennsymbol in N kann niemals ein weiteres Trennsymbol folgen.

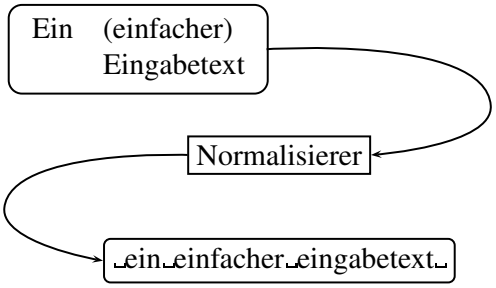


Abbildung 5.4.: Schematische Darstellung der Textnormalisierung

Beispiel 5.4.1. Der Text *„Ein (einfacher)\n\tEingabetext“*[5] werde mit dem Normalisierer bearbeitet, wobei die beiden Konfigurationsvariablen *ignore-case* und *ignore-punctuation* jeweils auf *wahr* gesetzt sind. Der Text wird in folgenden Schritten normalisiert (vgl. Abbildung 5.4):

1. Es wird das Trennsymbol vorne an den Eingabetext angefügt: *„␣Ein...“*
2. Es wird das Trennsymbol hinten an den Eingabetext angehängt: *„...Eingabetext␣“*.
3. Alle Punktuations- und Leerzeichen des Texts werden durch ␣ ersetzt; Alle Großbuchstaben im Text werden zu Kleinbuchstaben konvertiert: *„␣ein␣␣einfacher␣␣␣eingabetext␣“*
4. Alle nicht leeren Folgen von ␣ werden durch genau ein ␣ ersetzt: *„␣ein␣einfacher␣eingabetext␣“*

Suche auf den Texten

Die eigentliche Mustersuche wird unter Verwendung des $\mathcal{W}$-Automaten der Wissensressource, der ja aus den Lexikoneinträgen in $\mathcal{W}_{Lex}$ aufgebaut wird, auf den normalisierten Texten ausgeführt. Es wird, wie bereits erwähnt, eine *leftmost-longest* Strategie angewandt, welche immer die längst möglichen Muster, die am weitesten links im Text vorkommen, sucht. Es sollen immer möglichst lange Einträge gefunden werden. Auch sollen nur eindeutige Textbelege in den Texten gefunden

[5] $\backslash n$ steht für einen Zeilenumbruch und und $\backslash t$ für das Tabulator-Zeichen.

werden. Daher werden überlagernde Suchtreffer ignoriert. Sobald ein Lexikoneintrag im Text identifiziert wurde, kann daher der nächste Suchvorgang am Ende des Suchtreffers neu gestartet werden.

Da die Suche natürlichsprachliche Ausdrücke auf natürlichsprachlichen Texten finden soll und vor allem echte, natürlichsprachliche Textelemente in den Dokumenten identifiziert werden sollen, werden nur Suchtreffer akzeptiert, die von Tokentrennzeichen umgeben sind. Wie bereits in Kapitel 5.2.2 erwähnt, werden nicht nur die Eingabetexte der Suche normalisiert, sondern auch die Lexikoneinträge der Wissensressource. So wird auf einfache Weise sichergestellt, dass die gefundenen Textbelege immer von Tokentrennzeichen umgeben sind. Alle Lexikoneinträge sind immer von Tokentrennzeichen umgeben und sie können somit nur an solchen Positionen im Text gefunden werden, die ihrerseits wiederum von Tokentrennzeichen umgeben sind.

Wird ein Lexikoneintrag der Länge n in einem Text an den Positionen $p_i, p_{i+1}, \ldots, p_{i+n}$ gefunden, gilt somit immer $p_i = p_{i+n} = \text{␣}$. Für die entsprechende Textbelege, die gegebenenfalls in den Index eingetragen werden (vgl. Kapitel 5.3.2), werden diese beiden umgebenden Zeichen ignoriert. Es wird also nur der Textbeleg $p_{i+1}, \ldots, p_{i+n-1}$ in den Index geschrieben.

Die erneute Suche nach weiteren Textbelegen startet dann an der Position p_{i+n} im Text. Dies verletzt die Forderung, dass sich gefundene Textbelege nicht überlappen dürfen, sofern direkt ein neuer Textbeleg $p_{i+n}, \ldots, p_{i+n+o}$ gefunden wird (vgl. Kapitel 5.4.1). Da aber bei der Indexierung nur die Textbelege $p_i + 1, \ldots, p_{i+n-1}$ und $p_{i+n+1}, \ldots, p_{i+n+o-1}$ in den Index eingetragen werden, kann dieses Detail bei der Suche mit normalisierten Lexikoneinträgen auf normalisierten Texten ignoriert werden.

Beispiel 5.4.2. Die normalisierten Muster $P = \{\text{␣ab␣cd␣}, \text{␣}ab\text{␣}cd\text{␣}ab\text{␣}\}$ werden auf dem normalisierten Text „␣*cab*␣*cd*␣*ab*␣*cd*␣*ab*␣*cd*␣“ gesucht.

Die Suche liefert genau einen Treffer für das zweite Muster ␣*ab*␣*cd*␣*ab*␣, da das vorherige Vorkommen des ersten Musters *ab*␣*cd* nicht mit einem Trennzeichen beginnt und da beim zweiten Vorkommen des ersten Muster auch noch das längere Muster *ab*␣*cd*␣*ab* gefunden werden kann, von dem ersteres Präfix ist.

Das dritte Vorkommen des ersten Muster am Ende des Textes wird nicht gefunden, da einerseits sich überlappende Suchtreffer nicht zugelassen sind und da andererseits zuvor — das heißt weiter links im Text — bereits ein Muster gefunden wurde.

Die eigentliche Suche läuft auf dem normalisierten Text $N = n_1, n_2, \ldots, n_m$ ab. Eine Suche startet immer an einem Tokentrennzeichen[6] an der Position pos_{start}. Bei

[6]Da das erste Zeichen des normalisierten Textes *immer* das Trennsymbol ist, ist dies automatisch

der Suche werden die Buchstaben des Textes und die Zustände des $\mathcal{W}$-Automaten parallel durchlaufen. Kann ein Muster gefunden werden, das heißt, dass ein Finalzustand im Automaten erreicht werden konnte, startet eine neue Suche an der Position des Buchstabens, mit dem der Finalzustand erreicht wurde. Kann dagegen kein Muster von einer Startposition aus gefunden werden, das heißt, dass kein Finalzustand im Automaten erreicht werden konnte, wird eine erneute Suche vom nächsten Tokentrennzeichen nach der Startposition aus gestartet. Wurde das Ende des Textes erreicht, endet die Suche auf dem aktuellen Text. Dazu wird immer die aktuelle Position im Text *pos* und der aktuell aktive Zustand $q \in Q$ im $\mathcal{W}$-Automaten $= \langle \Sigma, Q, s, \delta, F, \theta, t \rangle$ der Wissensressource betrachtet.

Bei der Suche werden zwei weitere Hilfsvariablen verwendet. In der Variablen q_{last} wird der zuletzt gefundene Finalzustand im Lexikon abgelegt. In der Variablen pos_{last}, wird die Position gespeichert, von der aus der nächste Suchdurchlauf gestartet wird.

In einem Initialisierungsschritt wird $pos \leftarrow pos_{start}$ auf das erste Trennsymbol im Text gesetzt. Der aktive Zustand $q \leftarrow s$ wird auf den Startzustand s des $\mathcal{W}$-Automaten gesetzt. Die Variable $pos_{last} \leftarrow 0$ wird mit 0 initialisiert und $q_{last} \leftarrow \theta$ auf den Fallenzustand θ des Lexikonautomaten.

Solange $pos \leq m$ und $q \neq \theta$ gilt, werden folgende Schritte wiederholt:

1. Es wird ein Zustandsübergang mit dem Zeichen an der aktuellen Position durchgeführt: $q \leftarrow \delta(q, n_{pos})$.
2. Wenn $q \in F$: $q_{last} \leftarrow q$ und $pos_{last} \leftarrow pos$.
3. Wenn $n_{pos} = \text{␣}$ das Trennsymbol ist und wenn $pos \neq pos_{start}$ und $pos_{last} = 0$ gelten: $pos_{last} \leftarrow pos$.
4. Die aktuelle Textposition wird um eins erhöht: $pos \leftarrow pos + 1$.

Der Suchvorgang nach einem Muster des Lexikons endet, sobald entweder das Ende des normalisierten Texts erreicht ist, oder bis ein Zustandsübergang im Automaten zum Fallenzustand θ führt. Wenn dann $q_{last} \in F$ gilt, wurde bei der Suche ein Muster gefunden und q_{last} zeigt auf den letzten bei der Suche angetroffenen Finalzustand. Die entsprechende konzeptuelle Zuordnung kann direkt über die Typisierungsfunktion t des $\mathcal{W}$-Automaten erfolgen. Dazu muss in der Praxis lediglich die an der finalen Zustandszelle des Automaten gespeicherte Identifikationsnummer des Konzepts ausgelesen werden (vgl. Kapitel 3).

Das gefundene Muster liegt in diesem Fall zwischen pos_{start} und pos_{last} im Suchtext. Da die Suchmuster ihrerseits normalisiert sind, liegt das erste Zeichen des

beim Start der Suche gegeben.

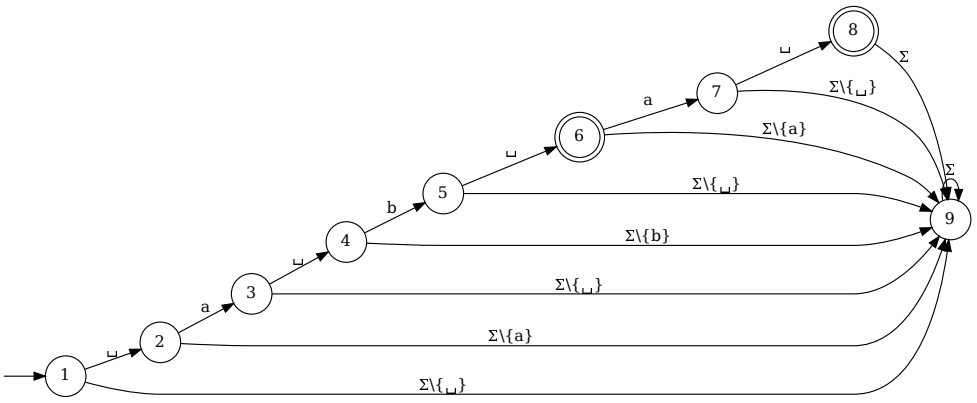


Abbildung 5.5.: Einfacher Lexikonautomat zur Erkennung der Suchmustermenge $P = \{␣a␣b␣, ␣a␣b␣a␣\}$. Zustand 9 markiert den Fallenzustand θ des Automaten, Zustände 6 und 8 dessen beide Finalzustände.

gefundenen Musters bei $pos_{start} + 1$, das letzte Zeichen an der Position $pos_{last} - 1$. Es gilt dabei immer $pos_{start} < pos_{last}$ und ebenso $n_{pos_{start}} = n_{pos_{last}} = ␣$. Da überlappende Suchtreffer ausgeschlossen werden sollen, kann, nachdem ein Muster im Text identifiziert werden konnte, die Suche im Text an der Position pos_{last} starten: $pos_{start} \leftarrow pos_{last}$.

Gilt am Ende der Suche dagegen immer noch $q_{last} = \theta$, wurde kein Finalzustand im Automaten erreicht und es konnte somit von pos_{start} aus kein valides Muster im Text gefunden werden. In diesem Fall ist in der Variable pos_{last} das erste Auftreten eines Trennsymbols während der Suche gespeichert. Wenn gilt: $pos_{last} \neq 0$ kann die erneute Suche nach einem Muster an der Position pos_{last} neu gestartet werden: $pos_{start} \leftarrow pos_{last}$.

Gilt dagegen $pos_{last} = 0$, wurde während der Suche kein Trennsymbol im Text angetroffen. In diesem Fall muss von pos_{start} aus das nächste Trennsymbol im Text gefunden werden. Die erneute Suche kann dann von dort aus neu gestartet werden.

Sobald das Ende des Textes erreicht wird, ist die Suche auf dem Dokument abgeschlossen und alle Muster im Text wurden gefunden.

Beispiel 5.4.3. Die beiden normalisierten Suchmuster $P = \{␣a␣b␣, ␣a␣b␣a␣\}$ werden auf dem normalisierten Text „$␣a␣a␣b␣a␣a␣a␣b␣$“ gesucht. Abbildung 5.5 zeigt den zur Suche nach den beiden Suchmustern verwendeten Lexikonautomaten als einfachen Trie. Konzeptuelle Zuordnungen sollen in diesem Beispiel vernachlässigt werden.

Die Suche startet mit $pos_{start} = 1$, $pos = 2$, $s = q_0 = 1$, $q_{last} = \theta = 9$ und

$pos_{last} = 0$. Der erste Suchschritt mit ␣ vom Startzustand aus ist erfolgreich. Es gelten somit $s = \delta(1, ␣) = 2$, $pos = 2$, $pos_{last} = 0$ und $q_{last} = 9$. Der nächste Suchschritt mit dem Buchstaben *a* ist ebenfalls erfolgreich und es gelten $s = 3$, $pos = 3$, $pos_{last} = 0$ und $q_{last} = 9$. Im nächsten Suchschritt mit dem Buchstaben ␣ wird nun erstmals ein weiteres Trennsymbol angetroffen. Da $pos_{last} = 0$ und $pos \neq 0$ gelten wird $pos_{last} \leftarrow pos$ gesetzt. Es gelten somit $s = 4$, $pos = 4$, $pos_{last} = 3$ und $q_{last} = \theta = 9$.

Der nächste Suchschritt vom Zustand 4 aus mit dem Buchstaben *a* scheitert nun und die aktuelle Suche wird abgebrochen. Da kein Finalzustand erreicht wurde gilt weiterhin $q_{last} = \theta = 9$. Bei dem letzten Suchlauf wurde ein Trennsymbol angetroffen, es gilt also $pos_{last} = 3 \neq 0$ und der nächste Suchdurchlauf kann an Position 3 starten.

Die nächste Suche startet an $pos_{start} = pos_{last} = 3$. Sie verläuft ähnlich wie die vorangegangene Suche und akzeptiert die Buchstabenfolge ␣, *a*, ␣, *b*, ␣. An Position $pos = 7$ wird erstmals der Finalzustand $s = 6$ im Lexikonautomaten angetroffen. Es werden daraufhin $q_{last} = 6$ und $pos_{last} = 7$ gesetzt und die Suche an $pos = 8$ fortgeführt.

Nach zwei weiteren Suchschritten wird an Position $pos = 9$ Finalzustand $s = 8$ angetroffen und $q_{last} = 8$ und $pos_{last} = 9$ gesetzt. Die Suche scheitert im nächsten Schritt an der Position $pos = 10$ im Text. Da $q_{last} \neq \theta$ gilt wurde ein Finalzustand im zwischen $pos_{start} = 3$ und $pos_{last} = 9$ gefunden. Über die konzeptuelle Zuordnung dieses Finalzustands kann der gefundene Textbeleg einfach seinem Konzept in der Wissensressource zugeordnet werden.

Da das Ende des Textes noch nicht erreicht worden ist, wird eine weitere Suche von $pos_{start} = pos_{last} = 9$ gestartet, die nach demselben Schema verläuft.

5.4.2. Indexerzeugung mit Wissensressourcen

Die Indexerzeugung mit einer Wissensressource $\mathcal{W}$ verwendet die Relationsmenge $\mathcal{W}_R$ und das Lexikon der Wissensressource $\mathcal{W}_{Lex}$, um die Dokumente der Dokumentenkollektion $d \in \mathcal{D}$ semantisch zu indexieren. Dabei wird ein semantischer Index aufgebaut, der, wie in Kapitel 5.3 dargestellt, aus einer Menge von Postingfiles besteht. Jedem direkt oder indirekt gefundenem Konzept der Wissensressource $k \in \mathcal{W}_K$ ist genau ein Postingfile zugeordnet.

Die einzelnen Textdokumente der Kollektion werden zunächst normalisiert. Nach der Textnormalisierung wird die multiple Mustersuche mit dem Lexikon auf dem normalisierten Text durchgeführt. Für jedes im Text gefundene Muster wird die konzeptuelle Zuordnung bestimmt und ein entsprechender Indexeintrag erzeugt, der dann, wie weiter oben bereits dargelegt, als direkter und indirekter Eintrag in die entsprechenden Postingfiles des semantischen Index geschrieben wird.

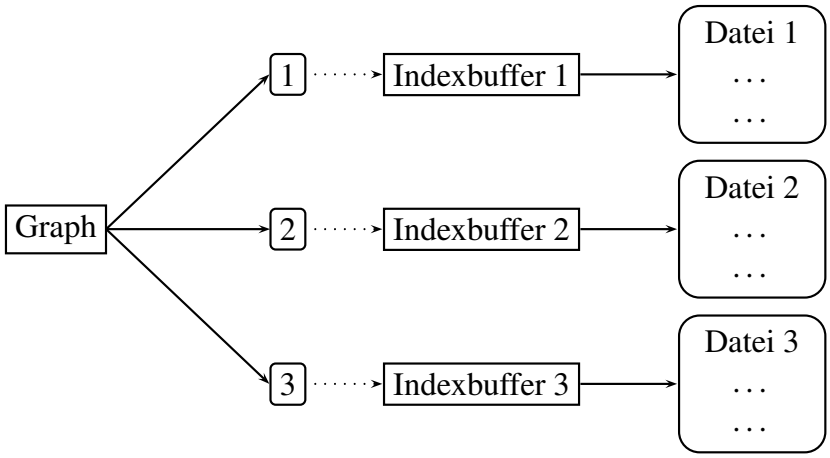


Abbildung 5.6.: Postingfiles und lokale Indexbuffer der Knoten.

Die benötigten Informationen zur Erzeugung der Indexeinträge stehen bereit, sobald ein Suchmuster im Text gefunden wurde. Die Textpositionen sowie das gefundene Muster können direkt über die pos_{start} und pos_{last} Variablen der Suche und den Text des Dokuments berechnet werden. Die Konzepte können aus den konzeptuellen Zuordnungen der Finalzustände des Automaten berechnet werden. Über die Kanten des Graphen, der zur Repräsentation der Relationsmenge der Wissensressource verwendet wird, stehen auch die indirekten Konzepte zur Verfügung. Der Dokumentenpfad des Indexeintrags ergibt sich direkt aus dem Pfad des aktuell bearbeiteten Dokument d.

Um die Menge der nötigen Lese- und Schreiboperationen auf den Postingfiles zu verringern, wird nicht jeder direkte und indirekte Indexeintrag sofort in das Postingfile geschrieben. Vielmehr werden die Indexeinträge in einem lokalen Buffer der Knoten zwischengespeichert und erst in die Postingfiles geschrieben, wenn sich eine bestimmte Anzahl von Indexeinträgen im Buffer angesammelt haben oder bis von externer Stelle das Schreiben aller Buffer erzwungen wird[7].

Um die Betrachtungen zum Aufbau des semantischen Index übersichtlicher zu halten, wird bei den folgenden Beschreibungen der Buffer der Knoten ignoriert und davon ausgegangen, dass jeder Indexeintrag eines Knotens direkt an das entsprechende Postingfile angehängt wird. In Wirklichkeit werden die Indexeinträge nicht einzeln sondern in Schüben in die Postingfiles geschrieben. Am Ende der

[7]Durch dieses Leeren der Indexbuffer muss die Indexierung auf einer Dokumentenkollektion abgeschlossen werden, da die Indexeinträge in den Puffern noch nicht für Anfragen an den Index sichtbar sind und erst sichtbar werden, sobald diese in die Postingfiles geschrieben werden.

Indexierung müssen gegebenenfalls noch halb gefüllte Indexbuffer an den Knoten explizit herausgeschrieben werden. Dieses ist in erster Linie ein Implementierungsdetail und hat keine Auswirkungen auf die allgemeinen Betrachtungen zur Indexerzeugung. Abbildung 5.6 zeigt zur Veranschaulichung den grundlegenden Aufbau der Knoten, deren lokale Indexbuffer und die zugehörigen Postingfiles.

Während die Suche auf den Dokumenten unabhängig von der verwendeten Wissensressource ist und für die Lexikoneinträge von EFGT-Netzen und Ontologien gleich abläuft, müssen abhängig von der verwendeten Wissensressource beim Aufbau des semantischen Index verschiedene Formen von Indexeinträgen in den Postingfiles abgespeichert werden. Je nachdem, ob die Wissensressource durch ein EFGT-Netz oder durch eine Ontologie repräsentiert ist, müssen verschiedene Indexeinträge in den Postingfiles verwaltet werden.

5.4.3. Erzeugung des Index aus EFGT-Netzen

Die Relationsmenge von EFGT-Netzen wie die in Kapitel 4.3.2 beschriebene TopicZoom Hierarchie verwendet im Gegensatz zu Ontologien nur eine einzige Relation $\mathcal{W}_K = \{R\}$. R ist dabei eine transitive, antisymmetrische Relation und die Verbindungen eines Konzepts zeigen immer auf all dessen übergeordnete Konzepte. Alle Verbindungen im Graphen stammen immer aus der gleichen Relation und sind somit gleich. Die Kanten des Graphen tragen keine Label. Jeder Knoten im Graphen kennt alle seine Oberthemen. Um die indirekten Konzepte eines Ursprungskonzepts zu finden, muss einfach nur über die ausgehenden Kanten des Knotens des Ursprungskonzepts iteriert werden.

Wie eingangs bereits erwähnt, wird für jedes direkt durch einen natürlichsprachlichen Ausdruck im Dokument gefundene Konzept immer ein direkter Indexeintrag erzeugt und in das Postingfile des Konzepts geschrieben. Zusätzlich dazu werden noch indirekte Indexeinträge für die Knoten erzeugt, mit denen dieser Ursprungsknoten verbunden ist. Bis auf die Information ob ein Indexeintrag direkt oder indirekt ist, enthalten alle Indexeinträge, die bei dieser Kaskade in den Index geschrieben werden dieselben Informationen.

Da in der Relation von EFGT-Netzen Unterthemen immer auf ihre Oberthemen zeigen, bzw. spezifischere Themen auf allgemeinere Themen zeigen und die allgemeineren Themen bei der semantischen Indexierung mit indexiert werden, können Anfragen nach allgemeineren Themen auch Dokumente liefern, die Vorkommen von spezifischen Unterbegriffen zu dem Thema enthalten.

Beispiel 5.4.4. Ausgehend von dem TopicZoom-Netz, das hier als Basis einer Wissensressource $\mathcal{W} = \langle \mathcal{W}_K, \mathcal{W}_R, \mathcal{W}_{Lex}, \kappa \rangle$ dient, soll hier der genaue Vorgang der

semantischen Indexierung dargestellt werden. Der Ausschnitt des Netzes ist in Abbildung 4.2 auf Seite 72 dargestellt.

In einem Dokument t_1 werde nun der Lexikoneintrag *„Sport"* $\in \mathcal{W}_{Lex}$ gefunden, der mit dem Konzept *Sport* des Netzes verbunden ist. Es wird nun ein direkter Indexeintrag in das Postingfile des Konzepts *Sport* eingetragen. Anschließend wird über alle ausgehenden Kanten des Konzepts *Sport* iteriert und ein indirekter Indexeintrag in das Postingfile des Zielknoten der Kante eingetragen. In diesem Fall wird nur ein indirekter Indexeintrag im Postingfile des Konzepts *Topnode* eingetragen, der als einziger von dem Konzept *Sport* referenziert wird.

In in einem anderen Dokument t_2 werden nun der natürlichsprachliche Ausdruck *Thomas Bach* $\in \mathcal{W}_{Lex}$ gefunden, der mit dem Konzept *Thomas Bach* des TopicZoom-Netzes verbunden ist. Es wird nun wiederum ein direkter Indexeintrag in das Postingfile des Konzepts *Thomas Bach* geschrieben.

Zusätzlich zu dem direkten Eintrag werden noch indirekte Indexeinträge für jeden Knoten erzeugt, der ein übergeordnetes Konzept repräsentiert und direkt von dem Ursprungskonzept referenziert wird: *Olympische Persönlichkeiten*, *Personen bei Organisationen und Einrichtungen*, *Organisationen, Institutionen und Einrichtungen*, *Persönlichkeiten des Bereichs Sport*, *Sport*, ..., *Topnode*.

Nach der Abarbeitung des Dokuments enthält das Postingfile des Knotens des Konzepts *Topnode* zwei indirekte Indexeinträge zu den Vorkommen von *Sport* und *Thomas Bach* in zwei unterschiedlichen Dokumenten t_1 und t_2. Das Postingfile des Knotens *Sport* enthält einen direkten Indexeintrag zu dem Vorkommen von *Sport* und einen indirekten Eintrag zu dem Vorkommen von *Thomas Bach*, wohingegen das Postingfile des Knoten von *Thomas Bach* nur einen direkten Eintrag zum Vorkommen von *Thomas Bach* im Dokument t_2 enthält. Alle anderen relevanten Postingfiles enthalten wie das Postingfile des Knotens von *Topnode* jeweils zwei indirekte Einträge.

5.4.4. Erzeugung des Index aus Ontologien

Grundsätzlich läuft die Indexerzeugung mit Ontologien ähnlich ab wie die Indexerzeugung mit EFGT-Netzen. Die Relationsmenge der Wissensressource der Ontologie umfasst mehrere Relationen mit unterschiedlichen, nicht weiter spezifizierten Eigenschaften. Sie ist als gerichteter Graph repräsentiert, der gelabelte Kanten besitzt. Für in Dokumenten gefundene Konzepte werden direkte Indexeinträge, für die referenzierten Konzepte indirekte Indexeinträge erzeugt.

Dabei werden die Ursprungsrelationen, die die Label der Kanten darstellen, unter denen ein indirekter Indexeintrag entstanden ist, mit in die indirekten Indexeinträge geschrieben. Hierzu werden die Identifikationsnummern $r_{id} > 0$ der Relationen verwendet. Direkte Indexeinträge werden mit einer 0 markiert.

Beispiel 5.4.5. Ausgehend von Abbildung 4.5 auf Seite 83, die die Verbindungen des Eintrags zu *Schlacht um Verdun* in der Ontologie zum ersten Weltkrieg darstellt, wird hier nun die semantische Indexierung mit Ontologien als Basis einer Wissensressource $\mathcal{W} = \langle \mathcal{W}_K, \mathcal{W}_R, \mathcal{W}_{Lex}, \kappa \rangle$ dargestellt.

Sei ein Dokument $w_1 \in \mathcal{D}$ der Dokumentenkollektion $w_1 =$*„Die Schlacht um Verdun begann am 21. Februar 1916.“*. Wenn nun w_1 indiziert wird, wird der Text zuerst normalisiert und dann nach natürlichsprachlichen Vorkommen der Konzepte durchsucht.

Der erste gefundene Textbeleg sei nun *„Schlacht um Verdun“* $\in \mathcal{W}_{Lex}$, welches dem Konzept *Schlacht um Verdun* zugeordnet ist. Es wird ein direkter Indexeintrag mit einer Identifikationsnummer von 0 erstellt, der auf die Positionen 5-22 im Dokument w_1 verweist und hinten an das Postingfile des Konzepts *Schlacht um Verdun* angehängt wird.

Im Gegensatz zu den indirekten Indexeinträgen des TopicZoom-Netzes, enthalten die jeweiligen Indexeinträge der indirekten Einträge die Identifikationsnummern $r_{id} > 0$ der Relationen der Wissensressource. Zur Erstellung der indirekten Indexeinträge wird der direkte Eintrag als Basis verwendet. Für jedes Konzept, das mit dem Konzept *Schlacht um Verdun* verbunden ist, wird die Identifikationsnummer basierend auf der Identifikationsnummer des Relationsnamen der Verbindung angepasst und an das Postingfile des entsprechenden Konzepts angehängt. Der direkte Indexeintrag enthält an Stelle der Identifikationsnummer eine 0, die diesen eindeutig als direkten Indexeintrag ausweist.

Jeder indirekte Indexeintrag enthält also zusätzlich den Namen der Relation, unter dem ein Eintrag entstanden ist. Es werden somit folgende indirekte Indexeinträge in die Postingfiles der Knoten zu den Konzepten *1916*, *Verdun*, *Philippe Pétain*, *Robert Nivelle*, *Erich von Falkenhayn*, *Frankreich* und *Deutschland* geschrieben. Es werden hier nur die direkt in der Ontologie ausgezeichneten Verbindungen zur Erzeugung indirekter Indexeinträge verwendet. So werden unter anderem keine weiteren indirekten Einträge für die mit dem Konzept *Erich von Falkenhayn* verbundenen Konzepte *Deutschland*, *1922*, *General*, usw. erzeugt.

Nachdem alle direkten und indirekten Indexeinträge geschrieben wurden, geht die Suche auf dem Dokument nach dem nächsten Lexikoneintrag weiter. Der nächste im Text identifizierte Lexikoneintrag ist der String *1916* an den Positionen 49-52. Es wird ein entsprechender direkter Indexeintrag erzeugt und in das Postingfile des Konzepts *1916* geschrieben. Das Konzept *1916* ist mit keinem weiteren Konzept verbunden und es werden keine indirekten Indexeinträge erzeugt. Es können keine weiteren Lexikoneinträge im Dokument gefunden werden und so endet die Indexierung von Dokument w_1.

Das nächste Dokument $w_2 \in \mathcal{W}$ enthält den Text *„Philippe Pétain war ein französischer General“*. Bei der Indexierung werden nun wiederum die Konzepte *Philippe*

Pétain („*Philippe Pétain*“), *General* („*General*“) und *Frankreich* („*französischer*“) im Text identifiziert und die entsprechenden direkten Indexeinträge in die Postingfiles geschrieben. Für die beiden Konzepte *Frankreich* und *General* werden keine weiteren indirekten Einträge erzeugt, da diese Konzepte über keine weiteren ausgehenden Verbindungen verfügen. Für das Konzept *Philippe Pétain* dagegen werden indirekte Indexeinträge in den Postingfiles der Konzepte *1856*, *1951*, *Frankreich* und *Military* erzeugt, die allesamt auf den Textbeleg „*Philippe Pétain*“ in Dokument w_2 verweisen.

Am Ende der Indexierung enthält der semantische Index eine Reihe von Postingfiles für alle direkt und indirekt in den Texten gefundenen Konzepten. So enthält zum Beispiel das Postingfile des Konzepts *Frankreich* einen direkten Eintrag, der auf den Textbeleg „*französisch*“ in Dokument w_2 verweist, einen indirekten Eintrag, der auf den Textbeleg in Dokument w_2 „*Philippe Pétain*“ mit dem Relationsnamen *Country* verweist und einen indirekten Eintrag, der auf den Textbeleg in Dokument w_1 „*Schlacht um Verdun*“ mit dem Relationsnamen *CountriesInvolved* verweist.

Der semantische Index umfasst auch Postingfiles von Konzepten, die niemals direkt in den Dokumenten $w_1, w_2 \in \mathcal{D}$ gefunden wurden. So enthält das Postingfile des Konzept *1951* einen indirekten Eintrag, der auf den Textbeleg „*Philippe Pétain*“ in Dokument w_2 verweist, obwohl das Konzept niemals selbst in einem der Dokumente identifiziert werden konnte.

Die Relationsmenge von Ontologien

Das Hauptunterscheidungsmerkmal von Ontologien zu EFGT-Netzen ist die Anzahl der Relationen der Relationsmenge der Wissensressource $\mathcal{W}_K$. Ontologien können mehr als eine Relation $\mathcal{W}_R = \{R_1, R_2, \ldots, R_n\}$ enthalten. Beim Aufbau des semantischen Index sind somit nicht nur direkte und indirekte Verbindungen von Bedeutung, sondern auch die Namen der Relationen unter denen die Konzepte untereinander verbunden sind.

Konzeptuell sind Ontologien und auch Thesauri (siehe Kapitel 4.7) oft aus einer Menge von Tripeln oder Termen aufgebaut. Das erste und dritte Element dieser Tripel definiert dabei meist die miteinander verbundenen Konzepte, das zweite Element den Namen oder das Prädikat der Verbindung (vgl. Kapitel 4.5). Die hier betrachteten Wissensressourcen bestehen nicht aus einer Menge von Tripeln sondern eben aus einer Menge von Relationen $\mathcal{W}_R$.

Diese beiden unterschiedlichen Betrachtungsweisen können leicht ineinander überführt werden. Die Tripel einer Ontologie können nach ihren Prädikaten gruppiert werden und in entsprechend benannte Relationen aus Subjekt- und Objekt-Paaren überführt werden. Genauso können die verschiedenen Relationen in $\mathcal{W}_R$

wiederum aus den Konzept-Paaren zu Tripeln mit den entsprechenden Prädikaten umgewandelt werden.

Beispiel 5.4.6. Der in Abbildung 5.1b dargestellte Graph einer Ontologie kann sowohl als eine Menge aus zwei unterschiedlichen Relationen $R = \{R_a, R_b\}; R_a = \{\langle A, B\rangle, \langle C, A\rangle\}, R_b = \{\langle B, C\rangle\}$ angesehen werden und andererseits als die einfache Tripelmenge $T = \{\langle A, a, B\rangle, \langle B, b, C\rangle, \langle C, a, A\rangle\}$.

Richtung der Indexierung

Neben der Anzahl der Relationen stellt die Richtung der Verbindungen einen weiteren Unterschied zwischen den Wissensressourcen von EFGT-Netzen und Ontologien dar. Bei der semantischen Indexierung werden immer indirekte Indexeinträge in Richtung der Verbindungen vererbt, so dass diese indirekten Einträge für Suchen nach den verbundenen Konzepten direkt bereitstehen.

Anders als bei EFGT-Netzen unterliegen die verschiedenen Verbindungen von Ontologien weniger strikten Regeln. So ist der Name der Relation, unter dem Konzepte einer Ontologie miteinander verbunden sind, oftmals wichtiger als die Richtung der Verbindung. Auch ist es keinesfalls immer so, dass nur speziellere Konzepte auf allgemeineren Konzepten zeigen.

Für die semantische Indexierung spielt die Richtung einer Verbindung dagegen eine Rolle. Speziellere Themen sind mit allgemeineren Themen verbunden und so werden bei der Indexierung die allgemeineren Themen gemeinsam mit den spezielleren Themen indexiert. Bei der semantischen Indexierung mit Ontologien müssen gegebenenfalls die einzelnen Relationen untersucht werden und die Richtungen der Verbindungen in Hinblick auf spätere Suchanfragen gegebenenfalls angepasst werden. Dazu können die Verbindungen einzelner Relationen $R_i \in \mathcal{W}_R$ einfach invertiert werden, indem beim Aufbau der Wissensressource die entsprechenden Umkehrrelationen R_i^{-1} erzeugt werden.

Beispiel 5.4.7. Eine Ontologie enthalte unter anderem auch Informationen zu deutschen Städten. Einträge zu Städten verweisen unter anderem über eine Relation *Stadtbezirk* auf eine Liste von Stadtbezirken der jeweiligen Städte[8], welche wiederum eigenständige Konzepte der Ontologie darstellen. So ist zum Beispiel in dieser Ontologie *München* über die Relation *Stadtbezirk* mit *Maxvorstadt* und *Schwabing-Freimann* verbunden.

Bei der semantischen Indexierung mit dieser Ontologie würden für jeden Textbeleg zu München immer auch indirekte Einträge zu den jeweiligen Stadtbezirken

[8] Eine solche Ontologie könnte auch automatisch aus Wikipedia-Artikeln erzeugt worden sein. So verweist der deutsche Wikipedia-Artikel zu München auf Wikipedia-Artikel zu Stadtbezirken und andere Sehenswürdigkeiten in und um München [64].

erzeugt werden und relationale Suchanfragen (vgl. Kapitel 5.5.3) zu Stadtbezirken würden auch Textbelege zu München zurückliefern.

In dieser Ontologie wird die Annahme der semantischen Indexierung verletzt, dass speziellere Konzepte immer auf allgemeinere Konzepte verweisen: Teile verweisen auf ihr Ganzes.

Falls diese Ontologie zur semantischen Indexierung verwendet werden soll, sollte die Relation *Stadtbezirk* invertiert werden, sodass Stadtbezirke auf die Stadt verweisen, in der sie liegen. Durch eine solche Invertierung der Relation *Stadtbezirk*, würden dann die Konzepte *Maxvorstadt* und *Schwabing-Freimann* über die Umkehrrelation $\textit{Stadtbezirk}^{-1} \approx \textit{ist-Stadtbezirk-von}$ mit dem Konzept *München* verbunden sein.

Bei der semantischen Indexierung mit dieser Ontologie werden dann für Textbelege zu verschiedenen Stadtbezirken jeweils auch indirekte Indexeinträge zu ihren verbundenen Städten erzeugt. Relationale Suchanfragen nach Städten liefern dann auch Textbelege zu Stadtbezirken, die in den angefragten Städten liegen. So würde eine relationale Suchanfrage nach *München* neben Textbelegen zu *München* nun auch Textbelege zu *Maxvorstadt* und *Schwabing-Freimann* zurückliefern.

5.5. Suchanfragen

Nach der Erzeugung des semantischen Index aus den Dokumenten einer Dokumentenkollektion, sobald alle Indexeinträge in ihre jeweiligen Postingfiles geschrieben wurden, kann der Index abgefragt werden. Die angefragten Elemente von Suchanfragen $?\,Q$ sind immer einzelne oder mehrere Konzepte der Wissensressource $Q \subseteq \mathcal{W}_K$.

Das Ergebnis von Suchanfragen an den semantischen Index ist eine Menge von Textbelegen aus den Postingfiles des Index. Die Textbelege referenzieren dabei immer Textstellen in Dokumenten, an denen ein entsprechender Lexikoneintrag der Wissensressource, der eines der angefragten Konzepte referenziert, gefunden wurde. Es können dabei durchaus auch mehrere Textbelege zum selben Dokument in der Ergebnismenge ausgegeben werden.

Die zentrale Einheit zur Bearbeitung von Suchanfragen stellen die jeweiligen Postingfiles der angefragten Konzepte dar. Einerseits können aus den Postingfiles die Textbelege zu einem angefragten Konzept direkt ausgelesen werden. Sie entsprechen genau den direkten Indexeinträgen in den Postingfiles der angefragten Konzepte. Andererseits können auch Textbelege zu Konzepten aus den Postingfiles der angefragten Konzepte ausgelesen werden, welche die angefragten Konzepten in irgendeiner der Relationen der Relationsmenge $\mathcal{W}_R$ der Wissensressource referenzieren. Diese Textbelege zu Konzepten, die ein angefragtes Konzept referenzie-

ren, sind über die indirekten Indexeinträge im Postingfile abgedeckt.

Da eben bei der semantischen Indexierung immer für jeden in einem Dokument identifizierten Textbeleg zu einem Konzept der Treffer einerseits direkt in das Postingfile des identifizierten Konzepts und andererseits auch in die Postingfiles der verbundenen Konzepte geschrieben wird, stehen die Textbelege zu untergeordneten bzw. referenzierten Konzepten immer auch in den Postingfiles der übergeordneten Konzepte. Die untergeordneten Konzepte eines angefragten Konzepts können somit immer direkt aus den Postingfiles der angefragten Konzepte ausgelesen werden.

So ermöglichen Suchanfragen auf einem semantischen Index nicht nur eine *einfache Suche* nach direkten Textbelegen zu Konzepten, sondern darüber hinaus auch eine *thematische* oder *relationale Suche*. Einfache Suchanfragen liefern dabei immer nur die *direkten Textbelege* in den Postingfiles der angefragten Konzepte zurück. Thematische Suchen dagegen liefern die *direkten und indirekten Textbelege* in den Postingfiles der angefragten Konzepte zurück. Der genaue Aufbau der verwendeten Wissensressourcen ist somit nur für relationale Suchanfragen, von Bedeutung nicht aber für einfache Suchanfragen.

5.5.1. Identifikation von Konzepten in Suchanfragen

Bei Anfragen an den semantischen Index müssen vor allem die angefragten Konzepte in der Suchanfrage identifiziert werden. Hierzu können zum einen die eindeutigen URIs der Konzepte verwendet werden und zum anderen wiederum die natürlichsprachlichen Lexikoneinträge in $\mathcal{W}_{Lex}$, welche die Konzepte ebenfalls eindeutig identifizieren. Die Identifikation von angefragten Konzepten in Suchanfragen kann dann auf eine ähnliche Weise durchgeführt werden wie die multiple Mustersuche der semantischen Indexierung.

Es ist offensichtlich, dass nur solche Konzepte mit natürlichsprachlichen Ausdrücken in Suchanfragen referenziert werden können, die auch über entsprechende Lexikoneinträge verfügen. Konzepte, denen keine Lexikoneinträge zugeordnet sind, können dagegen nur über ihre URI in der Suchanfrage identifiziert werden, wobei die Verwendung von Namensräumen oder die Verwendung von regulären Ausdrücken helfen können, die Handhabung der unhandlichen URIs in Suchanfragen zu erleichtern.

Ausdrücke zu Konzepten in Suchanfragen, die nicht auf ein Konzept der Wissensressource abgebildet werden können, müssen bei der Bearbeitung der Suchanfrage behandelt werden, indem diese Ausdrücke entweder ignoriert werden oder die Suchanfrage mit einem Fehler zurückgewiesen wird.

Beispiel 5.5.1. Das Konzept *Erich von Falkenhayn* der Ontologie „Erster Weltkrieg"

kann bei Suchanfragen entweder über seine natürlichsprachlichen Lexikoneinträge *„Erich von Falkenhayn"* und „Эрих фон Фалькенхайн" oder auch über seine volle URI `http://viaf.org/viaf/47021276` identifiziert werden.

5.5.2. Suchanfragen auf EFGT-Netzen

Die Relationsmenge einer Wissensressource von EFGT-Netzen verfügt nur über eine Relation $\mathcal{W}_R = \{R\}$. Dabei verbindet R immer Unterthemen mit ihren Oberthemen. Die Relation geht von semantisch engeren Konzepten hin zu semantisch weiteren. R ist transitiv abgeschlossen. Daher ist jedes Thema bzw. Konzept immer mit all seinen Oberthemen bzw. semantisch weiteren Konzepten verbunden.

Relationale Suchanfragen $?\,\mathcal{W}_R^{-1}(Q)$ nach einer *Anfragemenge* $Q \subseteq \mathcal{W}_K$ auf einem semantischen Index, der aus EFGT-Netzen mit der einzigen Relation $\mathcal{W}_R = \{R\}$ aufgebaut wurde, liefern einfach alle direkten und indirekten Textbelege in den Postingfiles der angefragten Konzepte $q \in Q$ zurück. Direkte Textbelege stellen tatsächliche Textbelege zu den angefragten Konzepten dar. Indirekten Textbelege dagegen sind konzeptuell Textbelege zu dem *Urbild* $R^{-1}(Q)$ der Anfragemenge Q unter der Relation R, die aufgrund der speziellen semantischen Indexierung direkt aus den Postingfiles der angefragten Konzepte ausgelesen werden kann.

Durch die strikte thematische Sortierung der Konzepte in EFGT-Netzen ergeben thematische Anfragen so Textbelege zu den angefragten Themen und all deren Unterthemen, welche allesamt in der Urbildmenge $R^{-1}(Q)$ der Anfragemenge enthalten sind.

Beispiel 5.5.2. Eine thematische Suchanfrage $?\,\mathcal{W}_R^{-1}(Q)$ mit der Anfragemenge $Q = \{$*Personen und Persönlichkeiten*$\}$ auf einem semantischen Index, der mit der auf Seite 72 in Abbildung 4.2 dargestellten TopicZoom-Hierarchie erzeugt wurde, liefert unter anderem Textbelege in Dokumenten zu den beiden Konzepten *Thomas Bach* und *Michael Bloomberg* zurück.

Zur Abarbeitung dieser Suchanfrage muss nur das Postingfile des Konzepts *Personen und Persönlichkeiten* eingelesen und alle direkten und indirekten Indexeinträge in dem Postingfile ausgegeben werden.

Die hierarchische Ordnung der Konzepte in EFGT-Netzen sowie die Transitivität der Relation führen dazu, dass Postingfiles von höherliegenden Konzepten mehr Indexeinträge ansammeln als tieferliegende. Das Konzept *Topnode* bildet die gemeinsame Wurzel aller Konzepte der Wissensressource und dessen Postingfile enthält immer *alle* Textbelege des semantischen Index.

Der semantische Index aus EFGT-Netzen ermöglicht es, mit einer Suchanfrage nach einem sehr abstrakten Konzept zu starten, interessante Textbelege zu ver-

schiedenen Unterthemen in der Ergebnismenge der Suchanfrage zu identifizieren und mit diesen Unterthemen weitere Suchanfragen an den semantischen Index zu stellen. Man kann so interaktiv von abstrakten Konzepten beginnend hin zu konkreteren, den gesamten semantischen Index nach interessanten Konzepten, Dokumenten und Textbelegen durchforsten.

5.5.3. Suchanfragen auf Ontologien

Im Gegensatz zu EFGT-Netzen, enthält die Relationsmenge von Ontologien zumeist mehrere, unterschiedliche Relationen. Das semantische Wissen, welches in Ontologien modelliert wird, ist nicht nur über die Konzepte und deren Verbindungen untereinander repräsentiert, sondern auch über die Relationen mit denen die Konzepte untereinander verbunden sind.

Aus genau diesem Grund werden bei der semantischen Indexierung mit Ontologien auch die Relationen von indirekten Indexeinträgen mit in den Index aufgenommen. Bei Suchanfragen kann so nicht nur zwischen direkten und indirekten Textbelegen unterschieden werden, sondern es kann auch auf den Namen der Relation zugegriffen werden, unter der ein indirekter Indexeintrag bei der Indexierung entstanden ist.

Allgemeine relationale Suchanfragen

Thematische Suchanfragen auf einem semantischen Index eines EFGT-Netzes müssen die Ursprungsrelation von indirekten Indexeinträgen nicht beachten, da diese immer implizit bekannt ist und Suchanfragen können daher einfach alle direkten und indirekten Indexeinträge in den entsprechenden Postingfiles zurückliefern. Ebenso liefern allgemeine, relationale Suchanfragen auf einem mit Ontologien erzeugten semantischen Index für eine Anfragemenge $Q \subseteq \mathcal{W}_K$ alle direkten und indirekten Indexeinträge in den Postingfiles der angefragten Konzepte $q \in Q$ zurück, ohne dabei die Ursprungsrelation der indirekten Indexeinträge zu beachten. Genau wie bei thematischen Suchanfragen werden einfach alle direkten und indirekten Indexeinträge in den Postingfiles der angefragten Konzepte zurückgeliefert.

Beispiel 5.5.3. Eine allgemeine relationale Suchanfrage $?\,\mathcal{W}_K^{-1}(Q)$ an einen semantischen Index, der aus der Ontologie aufgebaut ist, die auf Seite 83 dargestellt ist, mit einer Anfragemenge $Q = \{\textit{Deutschland}\}$, liefert alle direkten und indirekten Textbelege in dem Postingfile des Konzepts *Deutschland* zurück. Darunter auch Textbelege zu den Konzepten *Deutschland*, *Schlacht um Verdun* und *Erich von Falkenhayn*.

Wie aus dem vorangegangen Beispiel deutlich geworden ist, liefern allgemeine relationale Suchanfragen bei einem semantischen Index aus Ontologien eine Vielzahl von Textbelegen zu Themen zurück. Oft tauchen dabei auch Textbelege zu Konzepten auf, die intuitiv überhaupt nicht zur Suchanfrage passen. Dies liegt unter anderem auch an den weniger stark formalisierten Relationen und Verbindungen der Konzepte in Ontologien, bei denen die tatsächliche Relation, unter der Konzepte miteinander verbunden sind, eine weitaus wichtigere Rolle spielen, als die einfache Tatsache, dass zwei Konzepte miteinander verbunden sind.

Eingeschränkte, relationale Suchanfragen

Eingeschränkte, relationale Suchanfragen liefern im Gegensatz zu den allgemeinen relationalen Suchanfragen nicht mehr einfach alle direkten und indirekten Indexeinträge in den Postingfiles der angefragten Konzepte zurück. Um gezielter auf Ontologien suchen zu können und um die semantische Bedeutung der Relationen in der Ontologie mit in Suchanfragen einfließen zu lassen, können bei Suchanfragen auf einem semantischen Index einer Ontologie die zugelassen Ursprungsrelationen von indirekten Indexeinträgen gezielt eingeschränkt werden.

Hierzu können einfache relationale Suchanfragen auf einer Menge von Relationen $\mathcal{C} \subseteq \mathcal{W}_R$ der Relationsmenge der Wissensressource eingeschränkt werden. Solche auf $\mathcal{C}$ eingeschränkte, relationale Suchanfragen $?\,\mathcal{C}^{-1}(Q)$ liefern weiterhin alle direkte Textbelege in den Postingfiles der angefragten Konzepte $q \in Q \subseteq \mathcal{W}_R$ zurück, aber nur diejenigen indirekten Textbelege in den Postingfiles deren Ursprungsrelation Teil der *Einschränkungsmenge* $\mathcal{C}$ sind.

Falls für die Einschränkungsmenge $\mathcal{C} = \mathcal{W}_R$ gilt, handelt es sich bei der eingeschränkten relationalen Suchanfrage um eine allgemeine relationale Suchanfrage, da in diesem Fall alle möglichen Ursprungsrelationen von indirekten Indexeinträgen in den Postingfiles immer Teil der Einschränkungsmenge sind. Gilt dagegen $\mathcal{C} = \emptyset$, liefert die Suchanfrage nur die direkten Textbelege zu den angefragten Konzepten zurück, da keine der Ursprungsrelationen von indirekten Indexeinträgen Teil der leeren Einschränkungsmenge sein kann. Es handelt sich in diesem Fall um eine einfache Suche nach Textbelegen zu den Konzepten der Anfragemenge.

Die Einschränkungsmenge kann offensichtlicherweise nicht nur dazu verwendet werden, bestimmte Relationen bei Suchanfragen zuzulassen. Sie kann auch gezielt dazu eingesetzt werden, bestimmte Relationen $\mathcal{R} \subseteq \mathcal{W}_R$ aus Suchanfragen zu entfernen, indem die Einschränkungsmenge auf $\mathcal{C} = \mathcal{W}_R \setminus \mathcal{R}$ gesetzt wird.

Beispiel 5.5.4. Eine Suchanfrage $?\,\mathcal{C}^{-1}(Q)$ mit der Anfragemenge $Q = \{\textit{Deutschland}\}$ und einer Einschränkungsmenge $\mathcal{C} = \{CountriesInvolved\}$ liefert einerseits die direkten Textbelege zu dem Konzept *Deutschland* des Postingfiles zurück. Ande-

rerseits werden alle indirekten Textbelege zurückgeliefert, die über die Ursprungsrelation *CountriesInvolved* entstanden sind.

Da in der Ontologie „Erster Weltkrieg" über diese Relation nur Konzepte, die Ereignisse modellieren, mit Konzepten, die Länder modellieren, verbunden sind, liefert diese eingeschränkte, relationale Suche Textbelege zu *Deutschland* und Textbelege zu Ereignissen zurück, an denen *Deutschland* beteiligt war. Darunter auch Textbelege zu dem Konzept *Schlacht um Verdun*.

Die TopicZoom-Hierarchie als Ontologie

Die Wissensressource, der die TopicZoom-Hierarchie zugrunde liegt, enthält zwei zueinander inverse Relationen *broader* und *narrower*. Bei der semantischen Indexierung mit EFGT-Netzen wird die Relation *narrower* ignoriert und ist somit nicht Teil der Wissensressource, die zur semantischen Indexierung eingesetzt wird. Beide Relationen sind transitiv abgeschlossen und bilden eine Ordnung der Konzepte von Unterthemen hin zu Oberthemen und von Oberthemen hin zu Unterthemen.

Es ist aber durchaus möglich, beide Relationen bei der semantischen Indexierung zu verwenden. In diesem Fall wird die TopicZoom-Hierarchie bei der semantischen Indexierung und bei Suchanfragen wie eine Ontologie behandelt. Hierzu werden nun beide Relationen bei der Indexierung verwendet. Bei Suchanfragen kann dann durch die Verwendung von eingeschränkten, relationalen Suchanfragen eine der beiden Zuordnungsrichtungen für Suchanfragen verwendet werden.

Beispiel 5.5.5. Die TopicZoom-Hierarchie von Seite 83 werde zur semantischen Indexierung verwendet, wobei beide Relationen *narrower* und *broader* zur Indexierung eingesetzt werden. Für jedes in einem Dokument identifizierte Konzept werden neben den direkten Indexeinträgen die entsprechenden indirekten Indexeinträge für die Relation *broader* an die Oberthemen und für die Relation *narrower* an die Unterthemen weiter vererbt.

Auf die Relation *broader* eingeschränkte, relationale Suchanfragen liefern weiterhin Textbelege zu den angefragten Konzepten und zu deren Unterthemen zurück. Relationale Suchanfragen, die auf die inverse Relation *narrower* eingeschränkt werden, liefern dagegen Textbelege zu den angefragten Konzepten und deren *Oberthemen* zurück.

So liefert etwa eine Suchanfrage $?\,\{narrower\}^{-1}(\{Thomas\ Bach\})$ Textbelege zu der Person *Thomas Bach*, aber auch Textbelege zu dessen Oberthemen, wie zum Beispiel *Sport* zurück.

Dieses Vorgehen, zueinander inverse Relationen bei der semantischen Indexierung zu verwenden, ist nicht nur bei EFGT-Netzen möglich, sondern auch bei Ontologien, die zur semantischen Indexierung eingesetzt werden. Es kann für die je-

weiligen Relationen der Wissensressource zusätzlich auch deren inverse Relation mit bei der semantischen Indexierung verwendet werden.

Hierdurch kann sowohl der semantische Index angereichert werden als auch die Abfragemöglichkeiten zusätzlich erweitert werden. Die Hinzunahme weiterer Relationen vergrößert natürlich immer den resultierenden semantischen Index, da die Menge der zusätzlichen indirekten Indexeinträge, die für die in den Dokumenten identifizierten Konzepte erzeugt werden müssen, direkt von der Anzahl der verwendeten Relationen abhängt.

5.6. Hüllenbildung auf Relationen

Um alle transitiven Verbindungen der implizit transitiven Relationen der Netze zu erhalten, wird für EFGT-Netze die transitive Hülle der einzigen Relation immer automatisch berechnet. So wird die implizite Transitivität explizit ausgezeichnet und kann ohne spezielle Logik bei der semantischen Indexierung (Indexerzeugung und Indexanfragen) verwendet werden.

Bei Ontologien dagegen wurden bisher nur die explizit ausgeschriebenen Verbindungen in den Relationen der Relationsmenge betrachtet. Auch bei der semantischen Indexierung werden immer nur die explizit ausgeschriebenen Verbindungen der verschiedenen Relationen der Ontologie betrachtet. Ähnlich der Relation von EFGT-Netzen, verfügen oftmals Ontologien auch über implizit transitive Relationen. Ebenso sind einzelne Relationen in Ontologien denkbar, die implizit symmetrische Eigenschaften modellieren, ohne alle nötigen Verbindungen immer explizit anzugeben.

Bei solchen Relationen von Ontologien sind meist nicht alle Verbindungen explizit ausgeschrieben, da vor allem die Erhaltung von transitiven Verbindungen bei der manuellen Erstellung von Wissensbasen mühsam und fehleranfällig ist. Meist werden nur die direkten Verbindungen zwischen Konzepten behandelt und ausgezeichnet.

Um die implizite Transitivität oder Symmetrie einzelner Relationen von Ontologien bei der semantischen Indexierung explizit verwenden zu können, müssen die transitiven und symmetrischen Hüllen der entsprechenden Relationen berechnet werden. Die Berechnung geschieht wie die Berechnung der transitiven Hüllen der Relation von EFGT-Netzen auch einmalig bei der Erzeugung der Wissensressource zur semantischen Indexierung aus den Eingabedateien, indem die entsprechenden Hüllenbildungen gesondert für die jeweiligen Relationen durchgeführt werden. So stehen alle impliziten Verbindungen explizit bei der semantischen Indexierung zur Verfügung.

Die Eigenschaften der verschiedenen Relationen der Wissensressource müssen

dabei explizit ausgezeichnet sein. Das Programm zur semantischen Indexierung erlaubt es daher, verschiedene Eigenschaften von Relationen zu definieren, für die die entsprechenden symmetrischen oder transitiven Hüllenbildungen vorgenommen werden sollen.

5.7. Alternative Indexstrukturen

Neben der schon beschriebenen Struktur des Index sind weitere Indexarchitekturen für den semantischen Index möglich. Bisher ist im semantischen Index jedem Konzept der Wissensressource genau ein Postingfile zugeordnet, wobei die Dateinamen der Postingfiles eindeutig aus den konzeptuellen Zuordnungen der Konzepte hergeleitet werden. Alle Postingfiles liegen dabei in einer speziellen Verzeichnisstruktur, die den gesamten semantischen Index repräsentiert.

5.7.1. Gehashte Postingfiles

Ein Nachteil der einfachen Indexstruktur besteht darin, dass der Index gerade auch bei großen Wissensressourcen mit vielen Konzepten aus einer Vielzahl kleiner Postingfiles besteht. Gerade auch in virtualisierten Umgebungen sind unabhängig von dem zur Verfügung stehenden Festplattenspeicher die Menge der verfügbaren I-nodes — und damit die Menge der Dateien, die auf dem virtualisierten Rechner erzeugt werden können — stark beschränkt. Um die Menge der benötigten Postingfiles zu reduzieren, kann die einfache Implementierung des Index leicht erweitert werden.

Hierzu werden die Postingfiles mehrerer Konzepte mit einem geeigneten Hashing-Verfahren zusammengefasst. Das Hashing-Verfahren bildet die einzelnen Postingfiles der Konzepte auf eine feste Anzahl N von Postingfiles ab. Der Index ist bei diesem Schema eine Menge von Postingfiles $I = \{i_1, i_2, \ldots, i_N\}$ und das Postingfile eines Konzepts kann weiterhin eindeutig bestimmt werden. Die Postingfiles enthalten nun aber direkte und indirekte Indexeinträge zu mehreren unterschiedlichen Konzepten.

Da die Postingfiles bei diesem Verfahren Einträge zu unterschiedlichen Konzepten enthalten, muss die Zugehörigkeit eines Indexeintrags zu einem Konzept mit in den Index aufgenommen werden. Die Indexeinträge können weiterhin einfach ans Ende der Indexdateien angehängt werden, dabei entstehen wegen des Bufferings auf der Ebene der Knoten des Graphen Blöcke aus Indexeinträgen, die allesamt zu demselben Konzept gehören. Um nicht für jeden Indexeintrag sein zugehöriges Konzept speichern zu müssen, können für jeden Block zusätzlich Block-Header

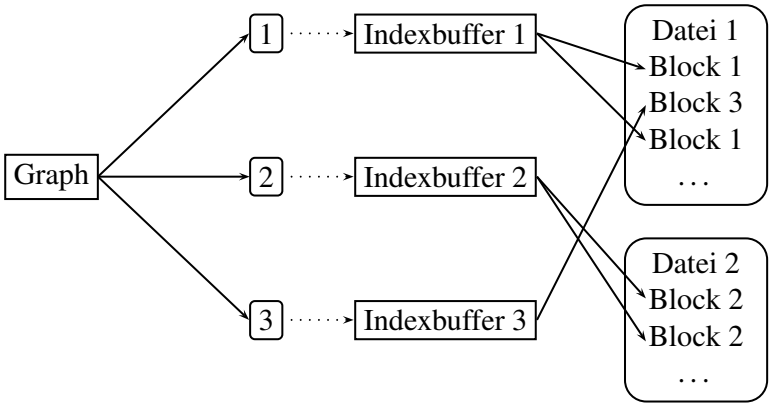


Abbildung 5.7.: Implementierung des Index mit gehashten Indexdateien

abgelegt werden, welche neben der Größe des folgenden Blocks auch die Zugehörigkeit der Indexeinträge enthält.

Bei der Abfrage des Index müssen die Postingfiles eingelesen werden, die wiederum über die Hash-Funktion und die angefragten Konzepte identifiziert werden. Das Lesen der Postingfiles geschieht dann blockweise, indem die Header der einzelnen Blöcke eingelesen werden. Dabei können Blöcke, die nicht zum angefragten Konzept gehören, direkt übersprungen werden und nur die Indexeinträge in den zugehörigen Blöcken beachtet werden.

Nachteilig bei der Verwendung dieses gehashten Indexaufbaus ist, dass die Anzahl N der zu verwendenden Postingfiles nur sehr schwer im Nachhinein verändert werden kann. Um die Obergrenze der Indexdateien in einem bestehenden Index zu verändern, müssen alle Blöcke aller Indexdateien neu auf Postingfiles verteilt werden.

Da die verwendete Hash-Funktion auf Basis der Konzepte arbeitet, kann es durchaus vorkommen, dass bestimmte Indexdateien überdurchschnittlich groß oder klein sind. Dies kann etwa geschehen, wenn mehrere sehr häufig oder sehr selten auftretende Konzepte in dieselbe Datei geschrieben werden müssen. Um dies zu verhindern, müsste die Hash-Funktion versuchen, die Konzepte auf der Basis ihrer Vorkommenswahrscheinlichkeiten zu verteilen. Es erscheint allerdings kaum möglich, dies rein auf der Basis von Konzepten zu erreichen. Ist die Wissensressource hierarchisch aufgebaut, wie dies in etwa bei EFGT-Netzen der Fall ist, ist es aber durchaus denkbar, gezielt mehrere, sehr spezielle Konzepte weit unten in der Hierarchie durch eine geeignete Hash-Funktion zusammenzufassen und sehr allgemei-

ne Konzepte sehr weit oben in der Hierarchie jeweils in ihren eigenen Postingfiles zu belassen.

5.7.2. Weitere Indexstrukturen

Neben den beiden einfachen, dateibasierten Indexierungsimplementierungen, sind noch weitere Implementierungen möglich. Dabei kann auf verschiedene *klassische* Verfahren zur Indexerzeugung zurückgegriffen werden.

Die Indextoken in den Indexbuffern an den Knoten des Graphen enthalten verschiedene Informationen, die allesamt in einfache relationale Datenbankschemata oder modernere nicht-rationale Datenbanken geschrieben werden können. Diese Datenbanksysteme verfügen über eine Vielzahl von weiteren Eigenschaften, die einerseits die Sicherheit und Wartbarkeit des Index erhöhen, andererseits den Index auch für externe Anwendungen benutzbar machen.

Da der eigentliche Index relativ wenig Anforderungen hat, kann der Index auch mit einem einfachen *Schlüssel-Wert Datenbanksystem* implementiert werden. Dazu eignet sich zum Beispiel die Berkley DB [42], welche ihrerseits oft zur Implementierung komplexerer Datenbanksysteme herangezogen wird. Dieses System ist dazu in der Lage, Daten als einfache Schlüssel-Wert Paare abzuspeichern. Der erzeugte Index kann dann effizient nach den Schlüsseln durchsucht werden. Im einfachsten Fall können die einzelnen Indexeinträge als Werte zu ihren entsprechenden Konzepten als Schlüssel abgespeichert werden und so einfach aus dem Index ausgelesen werden. Verschiedene Erweiterungen, wie weitere Indizes für die Relationen und Dokumente der Kollektion, sind möglich, um die Daten effizienter speichern und abfragen zu können.

Die unterliegende Datenbankimplementierung übernimmt dabei das Caching, die Konsistenzprüfung und das Abspeichern der eingelagerten Daten. Für die semantische Indexierung und die Abfrage des Index stehen aber weiterhin dieselbe Abstraktionsschicht bereit. Es stehen dieselben Möglichkeiten sowohl zur Speicherung des Index als auch zur Suche auf dem Index zur Verfügung.

5.8. Zusammenfassung

In diesem Kapitel wurde das grundlegende Verfahren zur semantischen Indexierung mit EFGT-Netzen und Ontologien dargestellt. Es wurde der allgemeine Aufbau des Index, die Erzeugung des Index aus natürlichsprachlichen Dokumenten und die Möglichkeiten von Suchanfragen an den semantischen Index erläutert.

Die Dokumente der Dokumentenkollektion, aus welcher der semantische Index aufgebaut werden soll, werden normalisiert und mit einem Lexikonautoma-

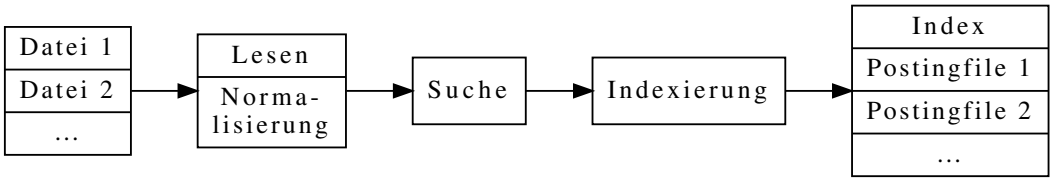


Abbildung 5.8.: Die möglichen Verarbeitungsschritte bei der semantischen Indexierung mit expliziten Wissensressourcen.

ten nach den natürlichsprachlichen Ausdrücken der Konzepte durchsucht. Diese Textbelege werden zum einen in den Postingfiles der gefundenen Konzepten abgespeichert. Zum anderen werden dieselben Textbelege als indirekte Einträge auch in den Postingfiles aller verbundener Konzepte abgelegt, welche durch das ursprüngliche Konzept im Graphen der Wissensressource referenziert werden.

Suchanfragen an den semantischen Index suchen immer nach Textbelegen zu den angefragten Konzepten in den Dokumenten. Bei den Suchanfragen kann das explizit in den Ressourcen gespeicherte Wissen über die Relationen der Wissensressource ausgenützt werden. Durch die Vererbung der Suchtreffer bei der Indexerzeugung ist die Urbildmenge von angefragten Konzepten direkt im Index hinterlegt und erlaubt die effiziente Bearbeitung thematischer Suchanfragen. Sofern die Relationen der verwendeten Wissensressource entsprechende Fakten abbilden, können Anfragen nach allgemeineren Themen so direkt auch Textbelege zu spezielleren, untergeordneten Themen zurückliefern.

Die Speicherung der Ursprungsrelationen bei der Vererbung von Indexeinträgen bei der Indexerzeugung, ermöglicht es, die verschiedenen Relationen von Ontologien bei den Suchanfragen mit zu berücksichtigen. Dies ermöglicht gezielte Suchanfragen, die verschiedene Fakten der Ontologie mit einbeziehen oder gezielt ausklammern.

Bei der Erzeugung eines semantischen Index spielt die Erkennung natürlichsprachlicher Ausdrücke eine zentrale Rolle. Somit spielen auch verschiedene sprachliche Phänomene in den Dokumenten eine Rolle. Die nächsten beiden Kapitel beschäftigen sich mit zwei sprachlichen Aspekten der Indexerzeugung: einerseits die Behandlung von Fehlern und historischen Schreibvarianten (Kapitel 6) und andererseits die Behandlung von sprachlichen Ambiguitäten (Kapitel 7).

Abbildung 5.8 stellt die Verarbeitungsschritte bei der semantischen Indexierung mit expliziten Wissensressourcen dar. Dieses Bild soll in den nächsten beiden Kapiteln noch verfeinert werden.

6. Behandlung historischer Schreibvarianten und Fehler

Viele Dokumente weisen Fehler aus verschiedenen Quellen auf, die sich negativ auf den Recall der semantischen Indexierung auswirken können, da unter Umständen Konzepte, die relevant für die semantischen Indexierung der Dokumente sind, aufgrund von Fehlern nicht korrekt in den Eingabedokumenten identifiziert werden können. In diesem Kapitel wird die Behandlung von Fehlern in den Eingabedokumenten der semantischen Indexierung mit expliziten Wissensressourcen dargestellt. Hierzu wird eine Form der unscharfen Suche mit dem Lexikonautomaten der semantischen Indexierung erläutert, die auch Vorkommen von Lexikoneinträgen mit Fehlern oder Schreibvarianten in den Dokumenten identifizieren kann. Ebenso werden die Auswirkungen der unscharfen Suche auf die semantische Indexierung beschrieben.

6.1. Fehler versus historische Schreibvarianten

Orthographische Fehler in Dokumenten sowie historische Schreibvarianten weisen eine Reihe von Gemeinsamkeiten auf. Wie bereits in Kapitel 2.6 erwähnt wurde, werden in dieser Arbeit Fehler und historische Schreibvarianten nicht weiter unterschieden sondern uniform behandelt.

Es sei hier aber nochmals explizit darauf hingewiesen, dass bestimmte Arten von Fehlern und vor allem auch historische Schreibvarianten durch Erweiterungen an den Einträgen in den Lexika der expliziten Wissensressourcen behandelt werden können. Darüber hinaus kann die hier dargestellte Behandlung von Fehlern und historischen Schreibvarianten in Dokumenten auf offensichtliche Weise mit diesen Erweiterungen der Lexikoneinträge kombiniert werden (vgl. Kapitel 2.6.2).

Gerade auch aus analogen Quellen digitalisierte Dokumente, die mit unterschiedlichen Techniken der optischen Zeichenerkennung oder *optical character recognition (OCR)* aus digitalisierten Bildquellen erzeugt wurden, enthalten die unterschiedlichsten Erkennungsfehler [52]. Im Falle von digitalisierten historischen Dokumenten können sich die Erkennungsfehler auch mit verschiedenen historischen

Abbildung 6.1.: Auszug einer digitalisierten Version von Castore Durantes *Hortulus Sanitatis* von 1609 (1609-H) [52]. Einige Probleme von digitalisierten historischen Texten wie etwa historische Schreibungen, Merges und Splits, die sich zusätzlich mit falsch erkannten Buchstaben überlagern, sind gut zu erkennen.

Schreibvarianten überlagern, welche die Behandlung der Texte mit modernen Lexika zusätzlich erschweren [24]. Es ist dabei zu beachten, dass im Allgemeinen Fehler nicht nur innerhalb einzelner Token auf der Basis *echter Zeichen*[1] auftreten, da gerade auch historische Dokumente oft auch schwer zu erkennende Tokengrenzen aufweisen [21]. Vielmehr kann schon die Tokenisierung selbst fehlerhaft erkannt worden sein, wenn Leerzeichen — und somit die hauptsächlichen Tokengrenzen — falsch erkannt wurden.

Nicht erkannte Leerzeichen führen zu verschmolzenen Token (*Merges*), überflüssig erkannte Leerzeichen zu zerschlagenen Einzeltoken (*Splits*). Erkennungsfehler der OCR können so nicht nur zu Fehlern in den erkannten Token führen, sondern auch zu falschen Tokenisierungen, die die Erkennung der Lexikoneinträge der Wissensressource erschweren. Neben solchen Erkennungsfehlern der OCR können auch Worttrennungen am Zeilenende zu falschen Tokenisierungen führen. Gerade auch in Hinblick auf die Mehrwortverbindungen in den Lexika der expliziten Wissensressourcen der semantischen Indexierung müssen diese Probleme der Tokenisierung bei der Indexierung fehlerbehafteter Dokumente behandelt werden. Abbildung 6.1 zeigt ein Beispiel einer fehlerbehafteten Zeichenerkennung und verdeutlicht die Probleme, die bei der Digitalisierung historischer Dokumente auftreten können.

[1] Gemeint ist hier die Menge aller darstellbarer Buchstaben ohne technische bedingte Metazeichen wie zum Beispiel Leerzeichen, die bei der optischen Zeichenerkennung mit betrachtet werden müssen.

6.2. Approximative Suche auf den Dokumenten

Um die semantische Indexierung auch auf fehlerbehafteten Dokumenten ausführen zu können, sollen auch fehlerbehaftete Textbelege zu Konzepten der Wissensressourcen in den Dokumenten identifiziert und indexiert werden können. Hierzu kann eine *fehlerbehaftete* oder *approximative* Suche auf den Dokumenten ausgeführt werden, die auch Textbelege findet, die einen bestimmten Levenshteinabstand $l \geq 0$ (vgl. Kapitel 1.4) zu Lexikoneinträgen der expliziten Wissensressourcen aufweisen.

Wie oben bereits erwähnt, treten Fehler nicht nur innerhalb von Einzeltoken, sondern auch an Tokengrenzen auf. Solche Splits und Merges erschweren die Suche nach Mehrwortverbindungen, da Fehler immer auch an Tokengrenzen zwischen Mehrwortverbindungen auftreten können. Bei der approximativen Suche müssen daher auch Fehler an Tokengrenzen behandelt werden. Hierzu werden bei der approximativen Suche Fehler an Tokengrenzen genau wie Fehler auf der Ebene der Buchstaben behandelt.

Beispiel 6.2.1. In dem normalisierten Text zu Abbildung 6.1 *...bewchrte␣artz␣ney␣...* kann ein Textbeleg zu dem ebenfalls normalisierten Lexikoneintrag *␣artznei␣* mit einem Levenshteinabstand von $l = 3$ gefunden werden.

Die historischen Schreibvarianten $z \rightarrow tz$ und $ei \rightarrow ey$ tragen genau wie auch der durch die Trennung am Zeilenende bedingte Tokenisierungsfehler zum gesamten Levenshteinabstand des Textbeleges zum Lexikoneintrag bei.

Während bei der exakten Suche (vgl. Kapitel 5.4.1) eindeutige Tokengrenzen angenommen werden können, sind wegen der Problematik von Merges und Splits auch die Anfangs- und Endpunkte bei der approximativen Suche nicht eindeutig. Fehler treten auch an Anfangs- und Endpunkten von Lexikoneinträgen auf. Solche Fehler müssen von der approximativen Suche behandelt werden.

Wie auch bei der exakten Suche sollen Suchtreffer natürlichsprachlichen Phrasen entsprechen und möglichst durch natürliche Tokengrenzen begrenzt sein. Ebenso wie bei der exakten Suche sind bei der approximativen Suche sich überlappende Suchtreffer nicht erlaubt. Es gilt somit auch für die approximative Suche, dass, nachdem ein Suchtreffer in einem Dokument gefunden wurde, eine neue Suche immer nach dem Ende des vorhergegangenen Suchtreffers starten muss. Für die Behandlung der Start- und Endpunkte bei der approximative Suche ergeben sich insgesamt vier Möglichkeiten:

1. Suchergebnisse müssen jeweils mit den Tokengrenzen des Textes übereinstimmen: sie müssen an einer Tokengrenze beginnen und an einer Tokengrenze enden.
2. Suchergebnisse müssen immer an Tokengrenzen beginnen, können aber innerhalb beliebiger Token enden.
3. Suchergebnisse können beliebig innerhalb von Tokengrenzen beginnen und enden.
4. Suchergebnisse müssen entweder hinter Tokengrenzen oder hinter vorherigen Suchtreffern beginnen und können auch innerhalb von Token beliebig enden.

Die erste Möglichkeit entspricht genau der exakten Suche. Bei der approximativen Suche können so Splits behandelt werden. Merges können jedoch nur innerhalb von Mehrwortverbindungen, nicht aber an den Anfangs- und Endpunkten von Suchtreffern.

Möglichkeit 2 erlaubt es zusätzlich noch auch Tokenisierungsfehler an den Endpunkten zu behandeln, wobei Suchtreffer, die innerhalb von Token enden, einen um eins erhöhten Levenshteinabstand erhalten. Tokenisierungsfehler am Anfang möglicher Suchtreffer werden allerdings nicht aufgelöst. Da die Suche weiterhin nur an Tokengrenzen beginnen darf, kann bei der Suche weiterhin tokenweise vorgegangen werden.

Im Gegensatz dazu erlaubt es Möglichkeit 3, eine größere Zahl möglicher Tokenisierungsfehler aufzulösen. Hierzu muss aber an jedem einzelnen Buchstaben des Textes eine neue approximative Suche gestartet werden. Dies führt zu einem deutlich größeren Aufwand bei der Suche. Zusätzlich dazu können, abhängig von den Lexikoneinträgen der expliziten Wissensressourcen, durch die fehlenden Synchronisationspunkte bei der Suche viele Affixe des Textes fälschlicherweise als Textbelege identifiziert werden.

Die letzte Möglichkeit stellt einen einfachen Kompromiss zwischen den Möglichkeiten 1 und 2. Suchtreffer beginnen entweder an Tokengrenzen oder direkt hinter vorangegangenen Suchtreffern, wobei Suchtreffer immer auch innerhalb von Token enden dürfen. So können Merges am Ende von Lexikoneinträgen und in bestimmten Fällen auch Merges am Anfang von Lexikoneinträgen behandelt werden.

Bei der semantischen Indexierung fehlerbehafteter Dokumente können grundsätzlich alle vier Möglichkeiten bei der approximativen Stringsuche angewendet werden. Da aber Möglichkeit 3 zu ineffizient ist, wird Möglichkeit 4 bei der hier dargestellten semantischen Indexierung als Kompromiss zwischen Effizienz und Sensitivität verwendet.

6.3. Approximative Suche auf dem Lexikonautomaten

Die Suche nach Lexikoneinträgen auf fehlerbehafteten Dokumenten kann als eine Instanz der *multiplen approximativen Patternsuche* bzw. der *unscharfen multiplen Patternsuche* mit einer konfigurierbaren Schranke $k \geq 0$ angesehen werden. Dabei gibt k die maximal zulässige Levenshteindistanz (vgl. Kapitel 1.4) zu irgendeinem Eintrag im Lexikon an. Diese Schranke k kann bei der semantischen Indexierung innerhalb *vernünftiger Grenzen*[2] frei gewählt werden, wobei die approximative Suche mit $k = 0$ genau der einfachen Suche aus Kapitel 5.4.1 entspricht.

Bei der approximativen Suche begrenzt k die im Text auffindbaren Lexikoneinträge. Je größer k ist, desto mehr mögliche Lexikoneinträge können in den Dokumenten gefunden werden. Dies kann unter Umständen sowohl zu falsch-positiven Indexeinträgen als auch zu mehrfachen unterschiedlichen konzeptuellen Zuordnungen einzelner Textpositionen führen. Bei der Wahl von k sollte somit vorsichtig vorgegangen werden, um die Menge der fehlerhaften Indexierungen zu minimieren. Wie im Folgenden noch dargestellt werden wird, können auch mehrere unscharfe Suchdurchläufe mit unterschiedlichen ansteigenden Schranken k_i mit einem exakten Suchdurchlauf kombiniert werden, um so auch die Menge der falschen Indexierungen zu minimieren.

Die hier dargestellte approximative Patternsuche verwendet dieselbe lexikalische Ressource $\mathcal{W}_{Lex}$ der Wissensressource $\mathcal{W}$ als Basis. Die unscharfe Suche bei der semantischen Indexierung braucht somit keine speziellen Ressourcen und kann ohne Erweiterungen auf der Dokumentenkollektion angewendet werden.

6.3.1. Nichtdeterministischer Automat zur approximativen Suche

In [14] wird ein Algorithmus beschrieben, der aus einem Eingabewort $w \in \Sigma^*$ und einer Fehlerschranke k einen nichtdeterministischen Automaten aufbaut, der alle Wörter $u \in \Sigma^*$ akzeptiert, für die gilt $L(u, w) \leq k$ (vgl. Kapitel 1.4.1).

Der Automat (vgl. Abbildung 1.2) hat ein starres Aussehen aus genau $k+1$ Ebenen und $(|w|+1)(k+1)$ Zuständen. Übergänge von einer Ebene k' in eine höher liegende Ebene $k'+1$ vergrößern immer auch die Levenshteindistanz des zu testenden Token u zu dem Eingabewort w. Es sind nur Sprünge von einer kleineren Ebene hin zu einer größeren möglich. Vertikale Zustandsübergänge im Automaten

[2] `semix` akzeptiert Schranken von $0 \leq k \leq 10$. Diese Beschränkung ist allerdings künstlich eingeführt um die verlängerte Laufzeit bei der approximativen Suche zu begrenzen.

entsprechen Einfügungen einzelner Zeichen, horizontale Zustandsübergänge entsprechen dem Akzeptieren eines Buchstabens ohne Fehler und diagonale Übergänge entsprechen Löschungen[3] oder Substitutionen auf Zeichenebene[4]. Akzeptiert der Automat ein Wort nicht, gilt: $L(u, w) > k$.

Dieser Automat ist vor allem dazu geeignet zu entscheiden, ob $L(u, w) \leq k$ gilt. Er eignet sich nicht zur multiplen approximativen Patternsuche nach Lexikoneinträgen, da zu diesem Zweck der Automat für *jeden* Lexikoneintrag erzeugt werden müsste und die Akzeptanz *aller* Automaten für *jedes* Token eines Eingabedokuments überprüft werden müsste[5]. Dieses Vorgehen ist gerade auch für große Lexika mit beliebig langen Einträgen unpraktikabel. Das Vorgehen wird hier nur erwähnt, da es die grundlegende Idee für die approximative multiple Patternsuche auf einem Lexikonautomaten liefert.

6.3.2. Berechnung aller möglichen Zustandsübergänge von Zuständen des Lexikonautomaten der Wissensressource

Der Algorithmus zur multiplen approximativen Patternsuche auf einem deterministischen Lexikonautomaten $A = \langle \Sigma, Q, s, \delta, F, \theta, t \rangle$ der expliziten Wissensressourcen benötigt für jeden Zustand des Automaten $q \in Q$ einerseits die Menge aller *Übergangszeichen* $\vec{\sigma}(q)$ und andererseits die Menge aller *Zielzustände* $\vec{q}(q)$.

Die Menge der Übergangszeichen eines Zustands des Automaten $\vec{\sigma}(q) := \{\sigma \in \Sigma | \delta(q, \sigma) \neq \theta\}$ bezeichnet genau die Menge aller Zeichen $\sigma \in \Sigma$ des Alphabets, mit denen ein Übergang von q mit σ aus ein Zustand im Automaten erreicht werden kann, der nicht der Fallenzustand θ ist.

Die Menge der Zielzustände eines Zustands des Automaten $\vec{q}(q) := \{\delta(q, \sigma) | \sigma \in \vec{\sigma}(q)\}$ ist die Menge aller Zustände des Automaten, die mit einem Übergangszeichen $\sigma \in \vec{\sigma}(q)$ aus erreicht werden und nicht der Fallenzustand sind.

Beide Mengen müssen bei der approximativen Suche für die verschiedenen Zustände des Automaten berechnet werden, wobei vor allem die Menge der Übergangszeichen eines Zustandes $\vec{\sigma}(q)$ wichtig ist, da die Menge der Zielzustände $\vec{q}(q)$ direkt aus $\vec{\sigma}(q)$ berechnet werden kann. Für kleine, eingeschränkte Alphabete können beide Mengen einfach berechnet werden, indem für jedes Zeichen

[3] Diagonale Übergänge, die mit dem leeren Wort ε markiert sind.

[4] Diagonale Übergänge, die mit Σ markiert sind.

[5] Eine andere Möglichkeit wäre, den Automaten für jedes Token des Eingabedokuments zu erzeugen und die Akzeptanz des Automaten für jeden Lexikoneintrag zu prüfen. Das oben dargestellte Vorgehen hätte allerdings zusätzlich den Vorteil, dass im Gegensatz zu dem hier skizzierten Vorgehen die Behandlung von Mehrwortverbindungen durchaus möglich ist.

des Alphabets der entsprechende Übergang explizit überprüft wird. Für größere, realistischere Alphabete ist dieses Vorgehen zu ineffektiv. Es müsste bei jeder Berechnung dieser Mengen über das *gesamte* Alphabet des Automaten iteriert werden[6].

Für die effiziente Implementierung der approximativen Suche müssen beide Mengen effizient für die jeweiligen Zustände berechenbar sein. Unabhängig von der Implementierung der Berechnung dieser Mengen können die Mengen für die jeweiligen Zustände in einer geeigneten Datenstruktur vorgehalten werden um überflüssige mehrfache Berechnungen zu verhindern.

In [47] werden die Übergangszeichen für die jeweiligen Zustände des Automaten in sogenannten *si-strings* gespeichert, aus denen dann die Menge der Zielzustände einfach für jeden Zustand berechnet werden kann. Das Lexikon, welches in `semix` implementiert ist, verwendet dagegen spezielle *Pointer* bzw. *Zeiger* in den Zellen der verschränkten Übergangstabelle (vgl. Kapitel 3.3.2) um die Speicherung von zusätzlichen si-strings zu vermeiden.

Jede nicht leere Zelle in der Übergangstabelle verfügt über einen solchen Pointer. Sie zeigen immer auf die nächste Übergangszelle eines Zustands. Ein Pointer $p = 0$ zeigt dabei immer an, dass entweder ein Zustand keine Übergänge hat oder dass keine weiteren Übergangszellen für den aktuellen Zustand im Automaten vorhanden sind. Die Zeiger müssen zusätzlich beim Aufbau der Übergangstabelle des Lexikonautomaten berechnet werden, sobald einzelne Zustände mit ihren Übergängen in die verschränkte Übergangstabelle des Automaten eingefügt werden[7].

Mit diesen Zeigern ist es möglich, iterativ für jeden beliebigen Zustand des Automaten gleichzeitig sowohl alle möglichen Übergangszeichen als auch deren entsprechende Zielzustände zu berechnen. Bei der approximativen Suche können so in einem Schritt sowohl alle Übergänge samt Zielzustände berechnet werden als auch direkt auf den Stack der approximativen Suche gelegt werden (vgl. Kapitel 6.3.3).

[6] Das Alphabet eines Lexikonautomaten kann gegebenenfalls sehr groß sein — so umfasst das Alphabet des Lexikonautomaten der Ontologie „Erster Weltkrieg“ (vgl. Kapitel 4.5.2) nicht nur das lateinische Alphabet mit den deutschen Umlauten, sondern auch Ziffern, verschiedene Spezialzeichen und kyrillische Zeichen zu Lexikoneinträgen in mindestens drei unterschiedlichen Sprachen.

[7] Zu diesem Zeitpunkt ist vom Algorithmus immer sichergestellt, dass keine weiteren Übergänge zu einem eingefügten Zustand hinzukommen — die Pointer der Übergangs- und Zustandszellen können sich daher nicht mehr ändern (vgl. Kapitel 3.3.2).

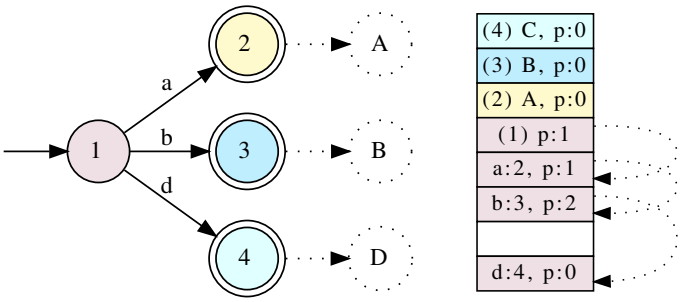


Abbildung 6.2.: Verschränkte Übergangstabelle eines Tries mit der Sprache $\mathcal{L} = \{a, b, d\}$ mit der Typisierungsfunktion $t = \{\langle a, A\rangle, \langle b, B\rangle, \langle d, D\rangle\}$. Abbildung eines einfachen Tries mit seiner zugehörigen verschränkten Übergangstabelle (vgl. Abbildung 3.3). Die Pfeile deuten die Ziele der Pointer in den Zellen an.

Beispiel 6.3.1. Abbildung 6.2 zeigt den Aufbau der verschränkten Übergangstabelle eines einfachen Lexikonautomaten. Die finalen Zustände q_2, q_3, q_4 des Automaten verfügen alle über keine ausgehenden Zustände. In den zugehörigen Zellen in der Übergangstabelle sind somit, neben ihren konzeptuellen Zuordnungen, die Pointer $p_2 = p_3 = p_4 = 0$ gespeichert.

Der Übergang 1 dagegen weist 3 mögliche, ausgehende Übergänge auf. Der Pointer der zugehörigen Zelle $p_1 = 1$ verweist auf die relative Position der ersten Übergangszelle. Die Pointer der jeweiligen Übergangszellen verweisen dann ihrerseits wieder auf die jeweils nächste Übergangszelle in der Tabelle. Schließlich deutet der Pointer $p_{1,d} = 0$ an, dass keine weiteren Übergänge für Zustand 1 existieren.

Um über alle ausgehenden Übergänge von Zustand 1 zu iterieren, kann so lange von Zelle zu Zelle gesprungen werden, bis eine Zelle mit einem Pointer $p = 0$ erreicht wird und somit keine weiteren Übergänge mehr vorhanden sind. Da in jeder der Übergangszellen sowohl das Zeichen des Übergangs und auch der Zielzustand gespeichert sind, sind immer sowohl das Übergangszeichen als auch der Zielzustand bekannt.

6.3.3. Nichtdeterministische Suche auf dem Lexikonautomaten der Wissensressource

Die Grundidee bei der multiplen approximativen Suche auf dem Lexikonautomaten ist, eine nichtdeterministische Suche auf dem deterministischen Lexikonautomaten zu implementieren. Dazu wird nicht mehr nur ein aktiver Zustand bei der

Suche betrachtet sondern eine Menge aktiver Zustände im Automaten. Die aktiven Zustände der Suche liegen dabei auf einem Stack. Es wird immer der oberste Zustand vom Stack genommen und verarbeitet. Neue aktive Zustände werden wiederum auf den Stack gelegt. Die neuen aktiven Zustände bei der nichtdeterministischen Suche auf dem Lexikonautomaten, die auf den Stack gelegt werden, orientieren sich dabei an dem nichtdeterministischen Levenshteinautomaten (vgl. Abbildung 1.2).

Wie der nichtdeterministische Levenshteinautomat verwendet die Suche eine Fehlerschranke k, die den maximalen Levenshteinabstand der akzeptierten Suchergebnisse zu irgendeinem Lexikoneintrag festlegt. Aus diesem Grund wird bei der Suche niemals ein Element auf den Stack gelegt, das einen aktiven Fehler $l > k$ aufweist. Dies folgt analog zu der festen Anzahl der Ebenen im nichtdeterministischen Levenshteinautomaten, die ebenfalls durch die Fehlerschranke k beschränkt sind. Bei der approximativen Suche müssen somit neben den aktiven Zuständen des Lexikonautomaten auch immer der aktuelle Fehler l des aktiven Zustandes mit beachtet werden. Da bei der Suche immer nur der Zustand oben auf dem Stack betrachtet wird und da auch einzelne Zeichen des Eingabetextes übersprungen werden können, muss auch immer noch das nächste zu lesende Zeichen des Eingabetextes beachtet werden.

Bei der approximativen Suche mit einer Fehlerschranke k und einem $\mathcal{W}$-Automat $A = \langle \Sigma, Q, s, \delta, F, \theta, t \rangle$ auf einem Eingabetext $T = t_1, t_2, \ldots, t_n \in \Sigma^*$, wird ein Stack verwendet, dessen Elemente Tupel $e = \langle q, l, t_i \rangle$ sind. Dabei heißt $q \in Q$ *aktiver Zustand*, $l \leq k$ *aktueller Fehler* des aktiven Zustandes und $t_i, i \leq n$ *nächstes Eingabezeichen*.

Bei der approximative Suche wird immer nur das oberste Element des Stacks $e = \langle q, l, t_i \rangle$ betrachtet. Dazu wird es von der Oberseite des Stacks entfernt und ein Zustandsübergang mit diesem Element durchgeführt. Neu entstandene, aktive Zustände werden wiederum mit samt ihrem aktuellen Fehler und dem nächsten Eingabezeichen auf den Stack gelegt.

Beim Start der Suche wird ein *Startelement* $e_0 = \langle s, 0, t_1 \rangle$ auf den Stack gelegt. Die Suche wird solange ausgeführt, bis der Stack leer ist, indem immer das oberste Element vom Stack heruntergenommen wird und dann die 4 möglichen Zustandsübergänge des nichtdeterministischen Levenshteinautomaten simuliert werden. Diese 4 Zustandsübergänge entsprechen den 4 Übergängen im nichtdeterministischen Levenshteinautomaten, die jeweils an dessen Zuständen möglich sind[8]:

[8] Diese 4 Übergänge sind nicht an allen Zuständen des nichtdeterministischen Levenshteinautomaten möglich. An solchen Zuständen wird durch einen aktiven Fehler $l > k$ verhindert, dass die unmöglichen Übergänge simuliert werden.

1. Horizontale Übergänge
2. Vertikale Übergänge
3. Diagonale Übergänge
4. ε-Übergänge

Im Folgenden werden nun die Übergänge des nichtdeterministischen Levenshteinautomaten genauer dargestellt. Bei den Darstellungen wird immer das aktive Stackelement $e = \langle q, l, t_i \rangle$ betrachtet, welches gerade von der Oberseite des Stacks entfernt wurde.

Horizontale Übergänge

Um einen horizontalen Übergang zu simulieren, wird vom aktiven Zustand des aktiven Stackelement aus ein Zustandsübergang im Lexikonautomaten mit dem nächsten Eingabezeichen durchgeführt. Wenn der Zielzustand des Übergangs nicht der Fallenzustand des Automaten ist und wenn $i + 1 \leq n$ gilt, wird ein neues Stackelement $e = \langle \delta(q, t_i), l, t_{i+1} \rangle$ auf den Stack gelegt.

Dies entspricht genau den horizontalen Übergängen des nichtdeterministischen Levenshteinautomaten. Wenn ein entsprechender Übergang ohne Fehler im Lexikonautomaten möglich ist, wird dieser eben auch durchgeführt. Dabei wird ein Zeichen des Eingabetextes konsumiert. Der Fehler wird dagegen nicht erhöht, da auch im nichtdeterministischen Levenshteinautomaten der horizontale Übergang in keine höher gelegene Ebene führt.

Vertikale Übergänge

Vertikale Übergang werden simuliert, indem ein neues Element $e = \langle q, l + 1, t_{i+1} \rangle$ auf den Stack gelegt wird, falls $l + 1 \leq k$ und $i + 1 \leq n$ gilt. Es wird somit das nächste Eingabezeichen konsumiert und der Fehler erhöht. Es wird aber *kein* Zustandsübergang im Lexikonautomaten durchgeführt.

Dies entspricht genau den vertikalen Übergängen im nichtdeterministischen Levenshteinautomaten, bei denen ein Zeichen des Eingabetextes konsumiert wird, aber kein Übergang in Richtung des zu erkennenden Worts des Automaten durchgeführt wird. Der Sprung in die höhere Ebene wird durch die Erhöhung des aktiven Fehlers nachgebildet.

Diagonale Übergänge

Die diagonalen Σ-Übergänge im nichtdeterministischen Levenshteinautomaten konsumieren ein Zeichen des Eingabetextes, erhöhen den Fehler und führen einen Übergang in Richtung des zu erkennenden Wortes mit einem beliebigen Zeichen des Alphabets durch. Da bei der approximativen Suche auf dem Lexikonautomaten nur Lexikoneinträge akzeptiert werden sollen, können nur Übergänge vom aktiven Zustand aus betrachtet werden, die mit Zeichen geschehen, die vom aktiven Zustand aus nicht im Fallenzustand des Automaten enden. Es werden also nicht die Übergänge mit allen Zeichen des Alphabets Σ betrachtet, sondern nur die Menge aller möglichen Übergangszeichen $\vec{q}(q)$ des aktiven Zustands q.

Zur Simulation der diagonalen Übergänge wird für jeden Zustand der Menge aller möglichen Zielzustände $q' \in \{\delta(q,\sigma)|\sigma \in \vec{q}(q)\}$ ein Stackelement $e' = \langle q', l+1, t_{i+1}\rangle$ auf den Stack gelegt, falls $l+1 \leq k$ und $i+1 \leq n$ gilt.

ε-Übergänge

Die diagonalen ε-Übergänge können auf eine ähnliche Weise wie die diagonalen Σ-Übergänge simuliert werden. Im Gegensatz zu diesen konsumieren sie allerdings *kein* Zeichen des Eingabetextes. Falls $l+1 \leq k$ gilt, wird für jeden der möglichen Zielzustände des aktiven Zustands $q' \in \{\delta(q,\sigma)|\sigma \in \vec{q}(q)\}$ ein neues Stackelement $e' = \langle q', l+1, t_i\rangle$ auf den Stack gelegt.

Anders als die anderen drei simulierten Übergänge des nichtdeterministischen Levenshteinautomaten werden die durch die ε-Übergänge referenzierten Zustände aktiviert, sobald ein neuer Zustand im nichtdeterministischen Levenshteinautomaten erreicht wird. Um dies bei der approximativen Suche nachzubilden, werden alle Elemente der ε-Übergänge auf den Stack gelegt, sobald ein neues Stackelement auf den Stack der approximativen Suche gelegt wird.

Alle Elemente — inklusive des Startelements e_0 — die während der approximativen Suche auf den Stack gelegt werden, erzwingen automatisch das Einfügen weiterer Elemente aus den ε-Übergängen. Diese zusätzlich auf den Stack gelegten Elemente können ihrerseits rekursiv das Einfügen weiterer Elemente erzwingen, bis der aktive Fehler der neu eingefügten Elemente größer ist als die Fehlerschranke k.

Finalzustände

Während der Suche müssen diejenigen Elemente des Stacks in der *Menge der finalen Positionen* S_F zwischengespeichert werden, deren aktiver Zustand final ist. Zum Beginn der approximativen Suche wird $S_F = \emptyset$ gesetzt. Wann immer bei

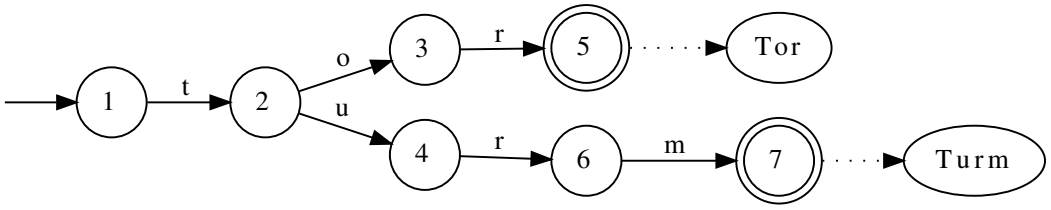


Abbildung 6.3.: Lexikonautomat zur approximativen Suche mit der Sprache $\mathcal{L}(A) = \{$*„turm"*, *„wurm"*$\}$

der Suche für den aktiven Zustand $e = \langle q, l, t_i \rangle$ $q \in F$ gilt, wird das zugehörige Element e zur Menge der finalen Positionen hinzugefügt: $S_F = S_F \cup \{e\}$.

Die approximative Suche endet, sobald der Stack leer ist und keine weiteren Elemente mehr verarbeitet werden können. Die Menge der finalen Positionen S_F enthält alle möglichen Elemente, die auf gefundene Textbelege zu Lexikoneinträgen von t_1 aus verweisen. Jedes Element $\langle q, l, t_i \rangle \in S_F$ verweist auf einen gefunden Textbeleg von t_1 nach t_i mit einem Fehler von l für ein Konzept $k = t(q)$[9]. Gilt dagegen $S_F = \emptyset$, konnten keine Textbelege zu Lexikoneinträgen mit einem Fehler $l \leq k$ von t_1 aus gefunden werden.

Beispiel 6.3.2. Mit dem in Abbildung 6.3 dargestellten Lexikonautomaten soll hier die grundsätzliche approximative Suche auf dem Text $T =$ *„torm"* mit einer Fehlerschranke von $k = 1$ dargestellt werden. Zur besseren Veranschaulichung markieren die Einträge des Stacks der Suche das nächste zu lesende Zeichen, indem das entsprechende Zeichen im Text unterstrichen dargestellt wird. Der eigentliche Algorithmus der approximativen Suche verwendet dagegen nur die Textpositionen des Eingabetextes.

Die Suche startet mit einem leeren Stack und $S_F = \emptyset$, indem das Startelement $e_0 = \langle 1, 0, \underline{t}orm \rangle$ auf den Stack gelegt wird. Die Einfügeoperation des Startelements erzwingt sofort das Einfügen weiterer Elemente der ε-Übergänge. Da $k = 1$ gilt wird nur ein weiteres Element $e_1 = \langle 2, 1, \underline{t}orm \rangle$ auf den Stack gelegt.

Im nächsten Schritt wird das gerade erwähnte Element e_1 wieder von Stack entfernt und für diese Element die verschiedenen Übergänge des nichtdeterministischen Levenshteinautomaten simuliert. Ein horizontaler Übergang kann nicht simuliert werden, da von Zustand 2 aus kein Übergang mit t möglich ist. Ebenso sind auch keine vertikalen oder diagonalen Übergänge möglich, da $l + 1 > k$ gilt. Es werden keine neuen Element auf den Stack gelegt.

Das nächste Element auf dem Stack ist das Startelement e_0. Es wird vom Stack heruntergenommen und es werden wiederum die verschiedenen Übergänge simu-

[9] t bezeichnet dabei die Typisierungsfunktion des KB-Automaten (vgl. Definition 1.5.3)

liert. Von e_0 aus ist ein horizontaler Übergang von Zustand 1 nach 2 möglich. Es wird das Element $e_3 = \langle 2, 0, \underline{t}orm\rangle$ des horizontalen Übergangs und die Elemente $e_4 = \langle 3, 1, \underline{t}orm\rangle$ und $e_5 = \langle 4, 1, \underline{t}orm\rangle$ der ε-Übergänge auf den Stack gelegt. Auch wird ein Element $e_6\langle 1, 1, \underline{t}orm\rangle$ des vertikalen Übergangs und ein Element $e_7 = \langle 2, 1, \underline{t}orm\rangle$ auf den Stack gelegt. Wegen der Fehlerschranke von $k = 1$ führen beide Elemente zu keinen weiteren Elementen durch die ε-Übergänge auf den Stack.

Als nächstes Element wird e_7 behandelt. Es wird nur ein neues Element $e_8 = \langle 3, 1, \underline{t}orm\rangle$ auf den Stack gelegt, da alle anderen simulierten Übergänge zu einem zu großen Fehler $l + 1 > k$ führen. Anschließend wird Element e_8 abgehandelt. Es wird wiederum nur ein Element $e_9 = \langle 5, 1, \underline{tor}m\rangle$ auf den Stack gelegt. Im nächsten Schritt wird e_9 behandelt. Da Zustand 5 final ist, wird e_9 der Menge der angetroffenen finalen Stackelemente S_F hinzugefügt. Die Suche hat den Lexikoneintrag *„tor"* mit einem Levenshteinabstand $l = 1$ zum Text gefunden. Da von Zustand 5 aus keine weiteren Übergänge mehr möglich sind und da der Fehler von e_9 bereits der Fehlerschranke k entspricht, können keine weiteren Elemente auf den Stack gelegt werden.

Das nächste Element der Suche auf dem Stack ist nun e_6. Da von Zustand 1 kein horizontaler Übergang möglich ist und da der Fehler bereits der Fehlerschranke k entspricht, werden keine neuen Elemente auf den Stack gelegt. Dasselbe gilt auch für die nächsten Elemente e_5 und e_4.

Bei der Abarbeitung von e_3 werden erstmals wieder neue Zustände auf den Stack gelegt. Darunter auch das Element $e_{10} = \langle 4, 1, \underline{t}orm\rangle$, welches schließlich über die horizontalen Übergänge dazu führt, dass die Elemente $e_{11} = \langle 6, 1, \underline{tor}m\rangle$ und $e_{12} = \langle 7, 1, torm\_\rangle$ auf den Stack gelegt werden und dass schließlich das finale Elemente e_{12} in S_F einfügt wird. Die Suche hat auch den Lexikoneintrag *„turm"* mit einem Levenshteinabstand $l = 1$ zum Text gefunden.

Die weitere Abarbeitung aller Elemente des Stacks folgt demselben Muster. Die Suche endet, sobald der Stack leer ist und keine weiteren Elemente mehr verarbeitet werden können. S_F enthält am Ende der approximativen Suche alle während der Suche angetroffenen Finalzustände, über die dann schließlich die Textbelege und die zugeordneten Konzepte der expliziten Wissensressource berechnet werden können.

6.3.4. Approximative Suche nach Mehrwortverbindungen

Wie die einfache Suche auch, startet die approximative Suche nach Mehrwortverbindungen *immer* an den Tokenbegrenzungszeichen des normalisierten Textes eines Dokuments. Dabei ist zu beachten, dass — wie auch bei der exakten Suche — die Lexikoneinträge genau wie der Eingabetext der Suche normalisiert sind (vgl.

Kapitel 5.4.1). Somit beginnen und enden *alle* Lexikoneinträge der expliziten Wissensressource immer mit *genau einem* Tokenbegrenzungszeichen „␣".

Anders als die exakte Suche, die wegen der normalisierten Lexikoneinträge automatisch immer an den Tokengrenzen beginnt und endet (vgl. Kapitel 5.4.1), muss bei der approximativen Suche auch die Möglichkeit von Textbelegen außerhalb der Tokengrenzen des Eingabetextes beachtet werden. Werden bei der approximativen Suche Lexikoneinträge — die ja immer auf ein Tokenbegrenzungszeichen enden — mit einem Fehler $l \leq k$ gefunden, können diese entweder an einer Tokengrenze des Textes oder aber auch innerhalb von Token enden. Dabei gilt es zu beachten, dass Textbelege, die innerhalb von Token eines Textes enden, aufgrund des abschließenden Tokenbegrenzungszeichen des Lexikoneintrags automatisch einen erhöhten Fehler aufweisen, da an solchen Endpositionen immer auch ein Substitutionsfehler zu dem Tokenbegrenzungszeichen erzwungen wird. Fehlerhaften Tokenisierungen an den Endpositionen von Lexikoneinträgen können somit automatisch wie alle anderen Zeichenfehler behandelt werden. Ebenso können auch falsche oder andersartige Tokenisierungen innerhalb von Mehrwortverbindungen als einfache Zeichenfehler behandelt werden.

Fehlerhafte Tokenisierungen können offensichtlicherweise nicht nur am Ende einzelner Textbelege auftreten, sondern auch an deren Anfang. Um alle möglichen Startpunkte von Textbelegen bei der approximativen Suche behandeln zu können, müsste eine approximative Suche nach Lexikoneinträgen von jedem Zeichen des Eingabetextes gestartet werden, wobei unter Umständen auch verschiedene Startpositionen innerhalb von Token miteinander verglichen werden müssten. Da dies aber zu ineffektiv wäre, wird mit einer neuen approximativen Suche immer nur an Tokengrenzen des Eingabedokuments begonnen. So kann der Text auch bei der approximativen Suche weiterhin auf der Ebene der Token durchwandert werden. In Einzelfällen können aus diesem Grund jedoch bestimmte Textbelege im Textdokument auch mit einer approximativen Suche nicht identifiziert werden.

Die approximative Suche startet somit immer von einer Textposition t_i aus, wobei immer $t_i = ␣$ gilt. Am Ende der approximativen Suche sind alle angetroffenen Finalzustände und damit auch die entsprechenden Endpositionen von Textbelegen zu Lexikoneinträgen von t_i aus in S_F enthalten. Dabei ist zu beachten, dass im Allgemeinen verschiedene Textbelege mit unterschiedlichen Endpositionen zu unterschiedlichen und gleichen Konzepten in S_F enthalten sein können. Die approximative Suche führt somit zu mehrdeutigen Textbelegen mit unterschiedlichen Endpositionen, die verschiedenen Konzepten der expliziten Wissensressource zugeordnet werden können.

Bei der approximativen Suche stellt sich somit die Frage, welche Konzeptzuweisungen verwendet werden sollen und von welcher Position aus eine erneute unscharfe Suche ausgeführt werden soll. Um mehrere Referenzen zu gleichen Kon-

zepten zu verhindern, werden mehrfache Vorkommen gleicher Konzepte aufgelöst, indem immer nur der längste Textbeleg zu einem Konzept mit dem geringsten Fehler beachtet wird. Andere mögliche Textbelege zu den gleichen Konzepten werden verworfen. So werden Textbelege mit einem möglichst geringen Fehler gefunden. Können mehrere Textbelege mit gleichem minimalem Fehler[10] gefunden werden, wird — der leftmost-longest Suchstrategie der strikten Suche folgend — der längste Textbeleg betrachtet.

Enthält S_F dann noch Textbelege zu unterschiedlichen Konzepten, können diese im Allgemeinen unterschiedliche Endpositionen — auch innerhalb von Token — aufweisen. Zur Berechnung der neuen Startposition der Suche werden nur die Konzeptzuweisungen verwendet, welche die längsten Textbelege aufweisen. Kürzere Konzeptzuweisungen zu andern Konzepten werden verworfen. Die neue Startposition ist dann die Endposition der Textbelege. Hierbei gilt zu beachten, dass immer zuerst der längste Textbeleg mit geringstem Fehler von *gleichen* Konzepten ausgewählt wird (sieh oben) und dann erst die längsten Textbelege zu *unterschiedlichen* Konzepten.

Dieses Vorgehen folgt dabei der leftmost-longest Suchstrategie der strikten Suche, wobei kürzere Textbelege zu weiteren Konzepten explizit verworfen werden. Diese Vorgehen ist allerdings nicht das einzig denkbare. Es ergeben sich weitere Vorgehensweisen zur Berechnung der Startposition:

1. Kürzere Textbelege werden nicht verworfen und die Suche startet an der Endposition des längsten Textbeleges.

2. Kürzere Textbelege werden nicht verworfen und Suche startet an der Endposition des kürzesten Textbeleges.

3. Es werden nur die Textbelege mit dem geringsten Fehler verwendet, wobei bei mehrfachen Textbelegen mit gleichem Fehler jeweils — wie bei den anderen beiden Vorgehensweisen — entweder die Endposition des längsten oder des kürzesten Textbeleges verwendet werden können.

Da der Levenshteinabstand nur wenig Aussagekraft über die Güte von Textbelegen liefert, erscheint Vorgehensweise 3 als am wenigsten geeignet, um die Startposition für einen erneuten Suchdurchlauf zu ermitteln. Die beiden anderen Möglichkeiten scheinen daher als die besseren Alternativen, die aber im Gegensatz zu dem in dieser Arbeit dargestellten Vorgehen, unter bestimmten Umständen noch mehr Konzeptzuweisungen für fehlerbehafteten Textbelege erzeugen können.

Nachdem die approximative Suche von einer Position aus abgehandelt wurde, wird eine neue Suche von der berechneten Startposition aus gestartet. Hierbei gilt

[10] Etwa zu unterschiedlichen Lexikoneinträgen des gleichen Konzepts.

zu beachten, dass die Lexikoneinträge der expliziten Wissensressourcen normalisiert sind und daher immer mit einem Tokenbegrenzungszeichen beginnen und enden. Da die Textbelege — auch wenn sie innerhalb von Token liegen — mit einem Begrenzungzeichen enden, kann der erneute Suchdurchlauf — wie auch bei der strikten Suche (vgl. Kapitel 5.4.1) — an der Endposition des vorher gefundenen Textbeleges starten. Falls die neue Suche dabei innerhalb eines Token startet, ist aufgrund der Normalisierung der Lexikoneinträge sichergestellt, dass automatisch ein entsprechend erhöhter Fehler bei der Suche berücksichtigt wird.

Werden dagegen keine Textbelege gefunden und gilt somit $S_F = \emptyset$, kann die Suche einfach am nächsten Tokenbegrenzungszeichen nach dem aktuellen starten.

Beispiel 6.3.3. Sei eine Wissensressource $\mathcal{W}$ mit einer Konzeptmenge $\mathcal{W}_K = \{\textit{Tor}, \textit{Turm}, \textit{Wurm}\}$, einem Lexikon $\mathcal{W}_{Lex} = \{\text{„␣}\textit{tor}\text{␣”}, \text{„␣}\textit{turm}\text{␣”}, \text{„␣}\textit{wurm}\text{␣”}\}$, den offensichtlichen Konzeptzuweisungen κ und einem entsprechenden $\mathcal{W}_K$-Automaten gegeben.

Eine approximative Suche auf dem normalisierten Text $T = \text{„␣}\textit{tnurin}\text{␣...”}$[11] mit einer Fehlerschranke von $k = 4$ liefert Textbelege zu *Tor* mit $l = 3$, zu *Turm* mit $l = 3$ und zu *Wurm* mit $l = 4$ zurück.

Der Textbeleg zu *Tor* endet nicht an der Tokengrenze des Eingabetextes sondern im Token selbst. Dieser Textbeleg wird ignoriert, da er kürzer ist als die Textbelege zu den anderen beiden Konzepten, die beide gleich lang sind und an einer Tokengrenze des Textes enden. Die approximative Suche liefert somit zwei unterschiedliche Konzepte mit jeweils einem Fehler $l = 3$ und $l = 4$ zu dem Textbeleg zwischen *„␣tnurin␣”*. Eine neue Suche kann jetzt direkt hinter dem gefundenen Textbeleg an dem Tokenbegrenzungszeichen starten.

Wären die anderen beiden Konzepte nicht gefunden worden, wäre das Konzept *Tor* das einzige identifizierte Konzept der approximativen Suche. In einem solchen Falle würde nur ein Konzept mit einem Fehler von $l = 3$ gefunden werden. Ein erneuter Suchdurchlauf würde dann an dem *i* in *„␣tnurin␣”* fortgeführt werden, wobei aufgrund der Normalisierung der Lexikoneinträge jeder Textbeleg mindestens einen Fehler von 1 aufweisen würde.

Es gilt noch zu beachten, dass bei der approximativen Suche beide Konzepte als Ergebnis zurückgeliefert werden. Die unterschiedlichen Levenshteinabstände werden nicht dazu verwendet, um zwischen multiplen gefundenen Konzepten auszuwählen.

Würde die hier verwendete Wissensressource noch das Konzept *Turin* mit dem normalisierten Lexikoneintrag *„␣turin␣”* enthalten, könnte an derselben Textstelle auch noch dieses Konzept mit einem Levenshteinabstand von $l = 1$ gefunden

[11] In diesem Beispiel sei die historische Form *„Thurm”* von einer OCR falsch erkannt worden — anstatt eines *h* wurde ein *n* erkannt und anstatt des Buchstabens *m* wurde die Folge *in* erkannt.

werden. In diesem Fall kann man ohne Kontext nicht mehr entscheiden, welches das beste dieser Konzepte zu diesem Textbeleg sein soll.

Die Levenshteindistanz dient somit vornehmlich der Filterung der möglichen Konzepte. Entstehende Ambiguitäten müssen immer an einer anderen Stelle in der semantischen Indexierung aufgelöst und behandelt werden.

6.3.5. Kombination mehrerer Suchen

Grundsätzlich ist die unscharfe Suche aufwendiger als die strikte Suche. Der bei der approximativen Suche verwendete Stack kann abhängig von den verwendeten Lexika und Fehlerschranken gegebenenfalls sehr groß werden. Es müssen daher viele aktive Zustände abgearbeitet werden. Da die verwendeten Lexikonautomaten konzeptuell Tries ähneln, die vor allem am Anfang viele mögliche Übergänge aufwiesen, müssen gerade beim Beginn von approximativen Suchen viele aktive Übergänge betrachtet werden.

Auch eignet sich die hier dargestellte approximative Suche nur bedingt dazu, strikte Textbelege[12] zu finden. Da die approximative Suche Ambiguitäten zwischen unterschiedlichen Konzepten niemals auf der Basis der Levenshteinabstände auflöst, können bei einer naiven Verwendung der approximativen Suche vermeidbare Ambiguitäten zwischen genauen und unscharfen Textbelegen auftreten.

Aus den oben genannten Gründen sollten approximative Suchen nur vorsichtig mit kleinen Fehlerschranken durchgeführt werden und sollten auch nicht dazu verwendet werden, strikte Textbelege ohne einen Fehler in den Dokumenten zu identifizieren. Es ist daher möglich, bei der semantischen Indexierung mehrere Suchen nacheinander auf demselben Text auszuführen. Hierzu können einer strikten Suche weitere approximative Suchen mit steigender Fehlerschranke folgen. Spätere Suchen werden dabei immer nur auf Teilen des Eingabetextes ausgeführt, auf denen vorherigen Suchen keine Ergebnisse lieferten.

- Strikte Suche wie bisher[13]
- Approximative Suche mit $k_1 > 0$
- Approximative Suche mit $k_2 > k_1$
- ...
- Approximative Suche mit $k_n > k_{n-1}$

[12] Gemeint sind hier Textbelege zu einem Lexikoneintrag mit einer Levenshteindistanz $l = 0$.

[13] Die strikte Suche ist streng genommen nicht zwingend, verhindert allerdings Ambiguitäten zwischen genauen und fehlerbehafteten Textbelegen.

6.3.6. Behandlung ambiger Suchergebnisse bei der unscharfen Suche

Die approximative Suche kann mehrfache unterschiedliche konzeptuelle Zuordnungen für einzelne Textbelege liefern (vgl. Beispiel 6.3.3). Diese *fehlerbehafteten ambigen Textbelege* haben einige Gemeinsamkeiten mit internen Ambiguitäten (vgl. Kapitel 2.4.3). Sie können jedoch bei der Suche selbst noch nicht aufgelöst werden.

Wie im nächsten Kapitel noch beschrieben wird, kann versucht werden die Ambiguitäten der unscharfen Suche genau wie sprachliche Ambiguitäten zu disambiguieren. Im Allgemeinen ist eine Disambiguierung aber nicht immer möglich oder wünschenswert. Ambige Textbelege der approximativen Suche können grundsätzlich ohne Disambiguierung indexiert werden. Hierzu werden einfach alle möglichen konzeptuellen Zuordnungen des Textbeleges indexiert. Die Ambiguität wird somit mit in den Index aufgenommen und bleibt auch für Indexanfragen bestehen.

Beispiel 6.3.4. Ein ambiger Textbeleg referenziere die Konzepte A und B. Bei der Indexierung werden für *beide* Konzepte A und B entsprechende direkte und indirekte Indexeinträge erzeugt, die jeweils den gleichen Textbeleg referenzieren. Der so indexierte Textbeleg wird somit sowohl bei Suchanfragen zum Konzept A als auch zu Suchanfragen nach Konzept B als Ergebnis zurückgeliefert.

Die semantische Indexierung geht davon aus, dass jedem Textbeleg genau ein Konzept zugeordnet ist, welches dann wie in Kapitel 5 indexiert und bei anschließenden Suchanfragen zurückgeliefert werden kann. Um die Ambiguitäten der approximativen Suche behandeln zu können, werden *anonyme Konzepte* verwendet. Diese anonymen Konzepte sind nicht Teil der Konzeptmenge der expliziten Wissensressource $\mathcal{W}_K$, sondern werden bei der approximativen Suche dynamisch erzeugt.

Wann immer ein fehlerbehafteter ambiger Textbeleg bei der unscharfen Suche auftaucht, wird ein temporäres anonymes Konzept X^* erzeugt, welches von dem gefundenen Textbeleg referenziert wird. Das anonyme Konzept referenziert dann seinerseits die verschiedenen möglichen Konzepte der expliziten Wissensressource. Anders als die einfachen Konzepte der expliziten Wissensressourcen tragen die Verbindungen zwischen dem anonymen Konzept und den Konzepten der Wissensressource nicht die jeweilige Relation sondern den zugehörigen Fehler, unter dem ein entsprechender String bei der unscharfen Suche gefunden wurde (vgl. Abbildung 6.4).

Die anonymen Konzepte werden bei der eigentlichen Indexierung gesondert behandelt, da ja nicht etwa das anonyme Konzept selbst[14], sondern die von diesem

[14]Da dieses anonyme Konzept dynamisch während der approximativen Suche erzeugt wird, ist

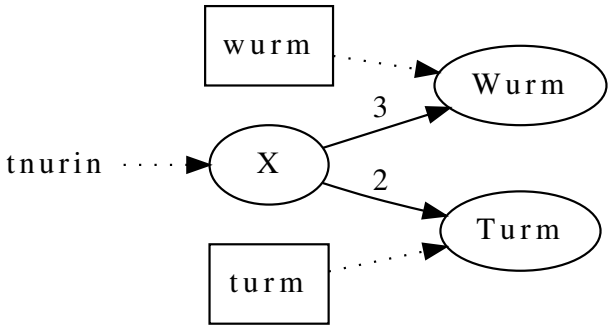


Abbildung 6.4.: Anonymes fehlerbehaftetes Konzept X^* einer approximativen Suche, welches von dem Textbeleg *„tnurin"* referenziert wird und selbst wiederum die Konzepte *Turm* und *Wurm* mit dem jeweiligen Fehler referenziert.

referenzierten Konzepte indexiert werden sollen. Bei der Indexerzeugung (vgl. Kapitel 6.5) werden daher niemals die anonymen Konzepte in den Index geschrieben, sondern die Konzepte, die von ihm referenziert werden.

6.4. Unscharfer Indexaufbau

Um bei späteren Indexanfragen auch auf die berechneten Levenshteinabstände zugreifen zu können und um zwischen exakten und unscharfen Textbelegen unterscheiden zu können, wird die berechnete Levenshteindistanz der Konzepte mit in den Index aufgenommen.

Da die maximalen Levenshteinabstände, die unter realistischen Umständen entstehen, sehr klein sind, muss kein neues Feld in den Indexeinträgen verwendet werden. Der Levenshteinabstand kann mit in das Feld der Indexeinträge geschrieben werden, welches angibt, ob ein Textbeleg direkt oder indirekt ist und unter welcher Relation der expliziten Wissensressource er zustande gekommen ist (vgl. Kapitel 5.3.1).

Bei EFGT-Netzen kann der Fehler von Einträgen im selben Byte abgelegt werden, in dem auch die Information darüber gespeichert ist, ob ein Indexeintrag direkt oder indirekt entstanden ist (vgl. Kapitel 5.3.1). Da in diesem Feld der Indexeinträge nur ein binärer Wert gespeichert wird, können die restlichen 7 Bits des Feldes zur Speicherung des Fehlers eines Indexeintrags verwendet werden. Dies

es eigentlich nicht Teil der expliziten Wissensressource. Es kann somit nicht bei der Suche referenziert werden. Ebenso besitzt es kein eindeutiges Postingfile.

ermöglicht es einen maximalen Fehler von $2^7 - 1 = 127$ in dem Feld abzuspeichern. Dies ist für praktische Anwendungen mehr als ausreichend.

Im Falle von Ontologien, die ja für die Indexeinträge nicht nur abspeichern, ob er direkt oder indirekt ist, sondern ebenso eine konzeptuelle Zuordnung der Indexeinträge in einem 32 Bit breiten Feld abspeichern (vgl. Kapitel 5.3.1), kann der Fehler ebenfalls in diesem Feld abgespeichert werden. Hierzu werden, ähnlich wie bei den Indexeinträgen von EFGT-Netzen auch, 7 Bit des 32 Bit breiten Feldes zur Speicherung des Fehlers des Indexeintrags reserviert. Dieses Vorgehen verkleinert zwar die maximale Anzahl der *unterschiedlichen* Relationen in der expliziten Wissensressource von 2^{32} auf $2^{32-7} = 2^{25} > 3 \times 10^7$, lässt aber für praktische Anwendungen immer noch genug unterschiedliche Relationen zu[15].

Der Aufbau der Indexeinträge von Ontologien und EFGT-Netzen mit der Berücksichtigung eines Fehlers unterscheidet sich somit nicht von der Darstellung aus Abbildung 5.2. Lediglich 7 Bit der jeweiligen Felder sind zur Speicherung des Levenshteinabstands reserviert. Die Indexeinträge sind miteinander kompatibel und können immer auf die gleiche Weise interpretiert werden. Die Speicherung mit oder ohne Fehler ist nur dann unterschiedlich, wenn tatsächlich ein Eintrag mit Fehlern in den Index geschrieben wird. Ansonsten sind die Fehler von Indexeinträgen 0.

6.5. Unscharfe Indexerzeugung

Bei der Erzeugung der Indexeinträge mit expliziten Wissensressourcen müssen drei unterschiedliche Arten von Textbelegen indexiert werden: Textbelege der strikten Suche, eindeutig fehlerbehaftete Textbelege und ambige fehlerbehaftete Textbelege. Textbelege der strikten Suche werden, wie in Kapitel 5.4 dargestellt, unter Verwendung eines Fehlers von $l = 0$ in den Index geschrieben. Für sie ergeben sich keine weiteren Besonderheiten bei der unscharfen Indexerzeugung.

Eindeutig fehlerbehaftete Textbelege und ambige fehlerbehaftete Textbelege können aus praktischen Gründen gleich behandelt werden. So muss bei der Indexerzeugung nur zwischen Textbelegen ohne Fehler und Textbelegen mit Fehlern unterschieden werden. Eindeutig fehlerbehaftete Textbelege, die nur ein Konzept der expliziten Wissensressourcen referenzieren, werden genau wie ambige fehlerbehaftete Textbelege auch von einem anonymen Konzept referenziert, welches den individuellen Fehler der entsprechenden Textbelege mitführt (vgl. Abbildung 6.4). Anonyme Konzepte von eindeutigen fehlerbehafteten Textbelegen referenzieren

[15] Hierbei gilt es auch zu beachten, dass nur die maximale Anzahl unterschiedlicher *Relationen* begrenzt wird — nicht aber die maximale Anzahl unterschiedlicher *Konzepte*.

genau ein Konzept — anonyme Konzepte ambiger fehlerbehafteter Textbelege referenzieren dagegen mehrere Konzepte.

Diese anonymen Konzepte müssen bei der Indexerzeugung anders behandelt werden als die einfachen Textbelege der exakten Suche. Es sollen bei der Indexierung ja alle möglichen konzeptuellen Zuordnungen mit berücksichtigt werden, da es im Allgemeinen nicht zu entscheiden ist, welches der Konzepte tatsächlich im Eingabetext referenziert wird. Anonyme Konzepte selbst erhalten dagegen keinen eigenen Indexeintrag — ihnen sind ja auch keine eigenen Postingfiles im Indexverzeichnis zugeordnet, da sie dynamisch erzeugt werden und nicht Teil der Wissensressource bei der semantischen Indexierung sind.

Die Konzepte, welche von einem anonymen Konzept referenziert werden, werden — wie Konzepte von strikten Suchen auch — als direkte und indirekte Indexeinträge in den Index geschrieben. Sowohl für direkte als auch für indirekte Indexeinträge wird immer auch noch der zugehörige Fehler mit gespeichert. Der Fehler von Textbelegen wird somit an die jeweiligen direkten und indirekten Konzepte vererbt und steht bei nachfolgenden Indexanfragen zur Verfügung.

Dieses Vorgehen bei der Indexerzeugung kann unter Umständen dazu führen, dass mehrere indirekte Einträge mit gleichen oder unterschiedlichen Fehlern zu demselben Textbeleg in den Index geschrieben werden. Dies passiert insbesondere dann, wenn mehrere Konzepte bei der unscharfen Indexerzeugung das gleiche Konzept referenzieren. Diese überflüssige Erzeugung mehrfacher Indexeinträge lässt sich leicht verhindern, indem für solche mehrfach referenzierten Konzepte nur ein Indexeintrag mit dem minimalen Fehler[16] aller beteiligten Konzepte erzeugt wird.

Beispiel 6.5.1. Das Ergebnis der approximativen Suche aus Beispiel 6.3.3 werde zur Erzeugung eines Index verwendet und das in Abbildung 6.4 dargestellte anonyme Konzept werde in den Index geschrieben.

Bei der Indexierung wird nun nicht das anonyme Konzept, sondern die beiden verbundenen Konzepte *Turm* mit einem Fehler von 2 und *Wurm* mit einem Fehler von 3 indexiert. Es werden zwei direkte Indexeinträge zu dem Textbeleg *„tnurin"* erzeugt: einmal im Postingfile des Konzepts *Wurm* mit einem Fehler von $l = 3$ und einmal im Postingfile des Konzepts *Turm* mit einem Fehler von $l = 2$.

Für jeden direkten Eintrag werden auch noch die entsprechenden indirekten Einträge (basierend auf der Wissensressource und den Verbindungen des Graphen) erzeugt. Diese indirekten Indexeinträge erben den jeweiligen Fehler ihrer direkten Einträge.

[16] Durch Speicherung des minimalen Fehlers kann der Textbeleg auch bei unscharfen Indexanfragen mit höherer Fehlerschranke als Ergebnis zurück geliefert werden (vgl. Kapitel 6.6).

Sei *Wurm* unter anderem mit dem Konzept *Lebewesen* und *Turm* mit dem Konzept *Bauwerk* verbunden, welche beide wiederum mit der Wurzel *Topnode* eines hypothetischen EFGT-Netzes verbunden sind. Es werden somit noch Indexeinträge zu den Konzepten *Bauwerk* mit einem Fehler von $l = 2$, *Lebewesen* mit einem Fehler von $l = 3$ und *ein* Indexeintrag zum Konzept *Topnode* mit einem Fehler von $l = 2$ erzeugt.

6.6. Unscharfe Indexanfragen

Im Allgemeinen müssen Indexeinträge mit einem Fehler $l > 0$ als potentiell *falsch* angesehen werden. Aus genau diesem Grund werden sie von normalen Suchanfragen ignoriert, indem sämtliche Indexeinträge in einem Postingfile, die einen Fehler $l > 0$ aufweisen, verworfen und nicht in die Ergebnismenge aufgenommen werden. Einfache Suchanfragen betrachten aus diesem Grund nur Indexeinträge mit einem Fehler $l = 0$ und liefern damit auch keine Textbelege zurück, die bei der fehlertoleranten Suche gefunden wurden.

Um explizit auch fehlerbehaftete Ergebnisse in Suchanfragen zuzulassen, können *fehlertolerante Suchanfragen* oder *unscharfe Suchanfragen* $?_k\, Q$ an den Index gestellt werden. Unscharfe Suchanfragen verwenden — ähnlich wie die approximative Suche — eine Fehlerschranke k, die den höchsten zulässigen Fehler in den Textbelegen der Ergebnismenge von Suchanfragen angibt.

Unscharfe Suchanfragen $?_k\, Q$ werden auf dieselbe Weise abgearbeitet wie normale Suchanfragen und es stehen die gleichen Suchmöglichkeiten zur Verfügung. Im Gegensatz zu einfachen Suchanfragen werden aber Indexeinträge mit einem Fehler von $l \leq k$ akzeptiert und nur Indexeinträge mit einem Fehler von $l > k$ bei der Abarbeitung ignoriert.

In Hinblick auf unscharfe Indexanfragen, stellen die einfachen Suchanfragen nur einen Spezialfall von unscharfen Suchanfragen dar. Normale Suchanfragen $?\, Q$ entsprechen unscharfen Suchanfragen mit einer Fehlerschranke von $k = 0$. Es gilt somit $?_0\, Q = ?\, Q$.

Beispiel 6.6.1. Eine Suchanfrage $?\, \{\textit{Turm}\}$ an den Index aus Beispiel 6.5.1 liefert keine Einträge für den Textbeleg zu *„tnurin"* zurück, da die entsprechenden Indexeinträge einen Fehler $l > 0$ aufweisen. Erst die Suchanfrage $?_2\, \{\textit{Turm}\}$ liefert den direkten Textbeleg *„tnurin"* zurück, der einen Fehler von $l = 2$ aufweist. Die Suchanfrage nach $?_2\, \{\textit{Wurm}\}$ liefert dagegen kein Ergebnis zurück. Erst $?_3\, \{\textit{Wurm}\}$ resultiert in dem Textbeleg zu *„tnurin"*.

Unscharfe Suchanfragen arbeiten auf dieselbe Weise wie einfache Suchanfragen und es stehen sämtliche Suchmöglichkeiten von normalen Suchanfragen zur Ver-

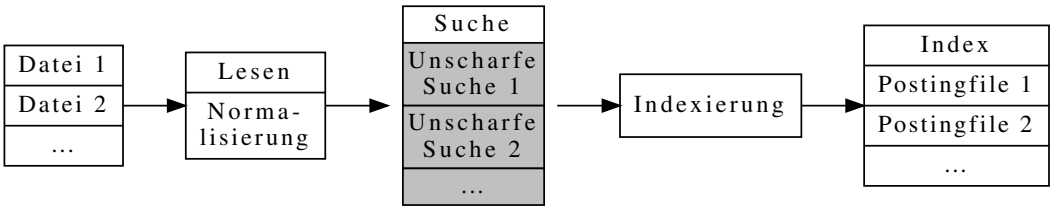


Abbildung 6.5.: Die möglichen Verarbeitungsschritte bei der semantischen Indexierung mit expliziten Wissensressourcen. Die optionalen Verarbeitungsschritte der unscharfen Suche sind grau hinterlegt.

fügung. So liefert $?\, R^{-1}(\{\textit{Lebewesen}\})$ wiederum keine Ergebnisse zurück, während $?_3\, R^{-1}(\{\textit{Lebewesen}\})$ den Textbeleg zu *„tnurin"* zurückliefert.

6.7. Zusammenfassung

Sollen historische und oder fehlerbehaftete Dokumente semantisch indexiert werden, können die auftretenden Abweichungen zwischen den Lexikoneinträgen auf der einen Seite und den Texten auf der anderen Seite durch nachgeschaltete approximative Suchschritte ausgeglichen werden. Die approximative Suche findet dann Lexikoneinträge, deren Levenshteinabstand kleiner oder gleich einer festgelegten Fehlerschranke k ist. Sie ist in eingeschränkter Weise auch dazu in der Lage, Mehrwortverbindungen über Tokengrenzen hinaus zu finden.

Wie üblich werden die gefundenen Lexikoneinträge ihren eindeutig zugeordneten Konzepten der expliziten Wissensressource zugeordnet und schließlich in den Index geschrieben, wobei der Levenshteinabstand der Suche an alle direkten und indirekten Indexeinträge vererbt wird. Dies ermöglicht es bei Suchanfragen auf den Levenshteinabstand der Textbelege zuzugreifen und die Ergebnisse zu filtern um entweder nur exakte oder aber auch vage Textbelege mit in die Ergebnisse aufzunehmen.

Abbildung 6.5 stellt die möglichen Verarbeitungsschritte bei der semantischen Indexierung mit einer fehlertoleranten Suche dar. Grundsätzlich können dabei der exakten Suche beliebig viele approximative Suchschritte mit aufsteigender Fehlerschranke nachgeschaltet werden, wobei alle Suchschritte immer vor der Indexierung erfolgen müssen.

Die approximative Suche kann zu Ambiguitäten führen. Diese Ambiguitäten werden explizit mit in den Index aufgenommen, da es nicht ohne weiteres möglich ist nur auf der Basis des Levenshteinabstands die *richtigen* Lexikoneinträge von den *falschen* zu unterscheiden. So können Indexeinträge unterschiedlicher Kon-

zepte auf gleiche Textbelege verweisen und bei unterschiedlichen Suchanfragen in die Ergebnismenge aufgenommen werden. Dies kann zur Zunahme von falsch-positiven Textbelegen führen, die aber weiterhin durch die Verwendung von Fehlerschranken bei den Suchanfragen eingeschränkt werden können.

Im nächsten Kapitel wird die Behandlung (sprachlicher) Ambiguitäten bei der semantischen Indexierung dargestellt. Die dort erläuterten Techniken können auch bei der Auflösung von Ambiguitäten, die aufgrund approximativer Suchen entstanden sind, helfen und die Menge der falsch-positiven Suchergebnisse weiter verkleinern.

7. Behandlung von Ambiguitäten

Die semantische Indexierung mit expliziten Wissensressourcen in der bisher dargestellten Form geht davon aus, dass ihren eindeutigen abstrakten Konzepten immer nur eindeutige natürlichsprachliche Ausdrücke zugeordnet sind. Wann immer ein Lexikoneintrag in einem Eingabedokument identifiziert werden kann, geht die bisherige semantische Indexierung davon aus, dass auch das entsprechende Konzept indexiert werden kann. Da aber in natürlichen Sprachen eine Vielzahl verschiedener Ambiguitäten bestehen, müssen Ambiguitäten bei der semantischen Indexierung behandelt werden. In diesem Kapitel werden drei unterschiedliche Aspekte von Ambiguitäten in Hinblick auf die semantische Indexierung behandelt. Als erstes wird die Behandlung ambiger natürlichsprachlicher Ausdrücke in den expliziten Wissensressourcen dargestellt. Darüber hinaus wird die Verarbeitung ambiger Textbelege bei der semantischen Indexierung erläutert. Als letztes werden noch verschiedene Formen der Auflösung von Ambiguitäten, die auf den verwendeten expliziten Wissensressourcen selbst basieren, behandelt.

7.1. Formen von Ambiguitäten

In dieser Arbeit werden nur Ambiguitäten auf Zeichenebene[1] behandelt. Ambiguitäten im Sinn dieser Arbeit sind somit Lexikoneinträge[2] in der Wissensressource, die nach einer Normalisierung (vgl. Kapitel 5.4.1) verschiedene semantische Bedeutungen innerhalb und außerhalb der Wissensressource aufweisen.

Beispiel 7.1.1. Das TopicZoom-Netz enthält unter anderem auch die beiden Konzepte *Gerhard Schröder(SPD)* und *Gerhard Schröder(CDU)*, die einerseits den SPD Politiker und anderseits den CDU Politiker mit dem Namen *Gerhard Schröder* referenzieren.

[1] Sogenannte *Homographen* — Wörter mit der selben Oberflächenform aber mit unterschiedlicher Bedeutung.

[2] Einzelwörter sowie Mehrwortverbindungen.

Dieser Name ist innerhalb des TopicZoom-Netzes ambig. Darüber hinaus gibt es auch außerhalb des Netzes weitere Personen mit diesem Namen. In Dokumenten ist daher *Gerhard Schröder* hochgradig ambig. Ohne weiteren Kontext in einem Eingabedokument ist nicht zu erkennen welche der beiden Personen referenziert werden soll und ob nicht eigentlich eine andere Person mit diesem Namen referenziert werden soll.

Wann immer ein normalisierter Lexikoneintrag $l \in \Sigma^*$ der Eingabedaten für die semantische Indexierung entweder manuell als Ambiguität ausgezeichnet ist oder wenn l als Lexikoneintrag für mehr als ein eindeutiges Konzept in den Eingabedaten dient, kommt es zu einer Ambiguität, die auf irgendeine Weise behandelt werden muss. Dabei wird dann l als *ambiger Lexikoneintrag* bezeichnet und die Menge all der Konzepte, die durch diesen ambigen Lexikoneintrag l referenziert werden, wird als Menge der *Mehrdeutigkeiten* oder als Menge der *an der Ambiguität beteiligten Konzepte* bezeichnet.

Wie bereits in Kapitel 2.4 dargestellt wurde, wird zwischen *internen* und *externen Ambiguitäten* unterschieden, deren Vorkommen immer von verschiedenen Faktoren, wie etwa der Domäne oder der Sprachstufe der Texte, abhängig ist. Interne Ambiguitäten stellen natürlichsprachliche Ausdrücke des Lexikons dar, die verschiedenen Konzepten der Wissensressource zugeordnet sind. Externe Ambiguitäten stellen dagegen Lexikoneinträge dar, die zwar eindeutig einem Konzept der Wissensressource zugeordnet werden können, außerhalb der Wissensressource aber weitere, nicht durch die Wissensressource abgedeckte Bedeutungen haben können. Wie Beispiel 7.1.1 zeigt, können sich interne und externe Ambiguitäten auch überlagern.

Beispiel 7.1.2. Das Lexikon einer Wissensressource enthalte den Lexikoneintrag *Bey*, der mit dem Konzept *Bey(türkischer Adelstitel)* verknüpft ist. Bei der Indexierung von modernen deutschsprachigen Texten ergeben sich mit diesem Lexikoneintrag kaum Probleme.

Will man nun historische deutsche Texte mit derselben Wissensressource indexieren und wird zusätzlich auch die Groß- und Kleinschreibung der Texte normalisiert, können nun plötzlich (externe) Ambiguitäten auftreten, da *bey* eine historische Schreibweise des modernen Worts *bei* ist. Es treten dann viele Textbelege zu dem Konzept *Bey(türkischer Adelstitel)* in Dokumenten auf und Suchanfragen nach diesem Konzept liefern somit viele irrelevante Dokumente zurück [20].

Grundsätzlich sollte schon bei der Erstellung der Lexika der Wissensressource darauf geachtet werden, möglichst wenige interne und externe Ambiguitäten in die Wissensressource einzuführen. Wenn möglich sollten ambige Lexikoneinträge

immer durch die Hinzunahme weiterer Kontextwörter aufgelöst werden (vgl. Kapitel 2.4.3). Darüber hinaus muss auch beachtet werden, dass durch die verwendete Textnormalisierung (vgl. Kapitel 5.4.1) zusätzliche Ambiguitäten in die Wissensressourcen eingeführt werden können.

7.2. Behandlung von Ambiguitäten in den expliziten Wissensressourcen

Ambiguitäten müssen bei der semantischen Indexierung schon beim Aufbau der Wissensressourcen behandelt werden. Eine wichtige Konsistenzbedingung der verwendeten Wissensressourcen (vgl. Definition 1.5.1) ist, dass alle Lexikoneinträge immer genau einem Konzept der Wissensressource zugeordnet sind. Interne Ambiguitäten verletzen diese Konsistenzbedingung, da sie mehreren Konzepten der Wissensressource zugeordnet werden können. Sie können beim Aufbau der expliziten Wissensressourcen automatisch erkannt und behandelt werden und müssen somit nicht speziell in den Wissensressourcen gekennzeichnet sein.

Externe Ambiguitäten dagegen verletzen die Annahme der semantischen Indexierung mit expliziten Wissensressourcen, dass Vorkommen von Lexikoneinträgen immer auch auf ein Vorkommen des verbundenen Konzepts im Eingabedokument hinweisen und so auch eindeutig in den Index übernommen werden können. Sie müssen in den Eingabedateien speziell ausgezeichnet sein, da diese aus offensichtlichen Gründen nicht beim Aufbau der expliziten Wissensressourcen automatisch erkannt werden können.

7.2.1. Behandlung externer Ambiguitäten

Externe Ambiguitäten betreffen Lexikoneinträge, die genau einem Konzept der Wissensressource zugeordnet sind. Im Allgemeinen sind dem gleichen Konzept auch noch andere nicht-ambige Lexikoneinträge zugeordnet. Die Markierung ambiger Lexikoneinträge muss somit auf der Basis einzelner Lexikoneinträge geschehen, da Konzepte sowohl von ambigen als auch von eindeutigen Lexikoneinträgen referenziert werden können.

Um externe Ambiguitäten zu behandeln, werden für alle als ambig bezeichneten Lexikoneinträge, die ein Konzept $k \in \mathcal{W}_K$ der Wissensressource $\mathcal{W}$ referenzieren, ein neues *ambiges Konzept* k^* in $\mathcal{W}_K$ eingefügt. Der ambige Lexikoneintrag referenziert dann nicht mehr das ursprüngliches Konzept k sondern das ambige Konzept k^*. Über eine speziell ausgezeichnete Relation $\mathrm{Amb} \in \mathcal{W}_R$ wird k^* mit dem ursprünglichen Konzept k verbunden: $\mathrm{Amb}(k^*, k)$. Andere nicht ambige Le-

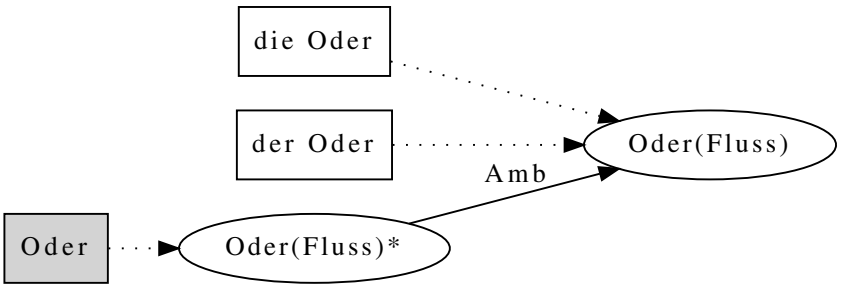


Abbildung 7.1.: Repräsentation externer Ambiguitäten. Der ambige Lexikoneintrag *„Oder"* referenziert das Konzept *Oder(Fluss)* indirekt über das ambige Konzept *Oder(Fluss)**. Die eindeutigen Lexikoneinträge *„die Oder"* und *„der Oder"* referenzieren das Konzept *Oder(Fluss)* dagegen weiterhin direkt.

xikoneinträge des eindeutigen Konzepts k dagegen, referenzieren weiterhin das ursprüngliche Konzept k (vgl. Abbildung 7.1).

Grundsätzlich würde es für die Behandlung von externen Ambiguitäten ausreichen, die entsprechenden Lexikoneinträge mit einer einfachen Markierung auszustatten, anstatt ein neues, ambiges Konzept in die Wissensressource einzuführen. Die hier gewählte Behandlung externer Ambiguitäten ist aber synonym zu der Behandlung interner Ambiguitäten (siehe unten). So wird eine uniforme Behandlung externer und interner Ambiguitäten bei der semantischen Indexierung ermöglicht. Darüber hinaus erleichtert dieses Vorgehen beim Aufbau der Wissensressourcen die Behandlung sich überlagernder externer und interner Ambiguitäten.

7.2.2. Behandlung interner Ambiguitäten

Interne Ambiguitäten betreffen mehrdeutige Lexikoneinträge, die mehrere Konzepte der Wissensressourcen referenzieren. Anders als externe Ambiguitäten müssen sie nicht explizit in den Eingabedateien ausgezeichnet sein, sondern können automatisch erkannt werden.

Eine interne Ambiguität liegt immer genau dann vor, wenn in den Eingabedaten, die zum Aufbau der Wissensressourcen verwendet werden, ein bestimmter Lexikoneintrag mehr als ein Konzept der Wissensressource $k_1, \ldots, k_n \in \mathcal{W}_K; n > 1$ referenziert. Da solch ein Fall in den expliziten Wissensressourcen der semantischen Indexierung aber nicht erlaubt wird (vgl. Definition 1.5.1), müssen interne Ambiguitäten immer auf eine geeignete Weise behandelt werden.

Grundsätzlich sind drei unterschiedliche Vorgehensweisen zur Behandlung von ambigen Lexikoneinträgen denkbar. Im einfachsten Fall können entsprechende

ambige Lexikoneinträge verworfen werden. Sie werden dann nicht in das Lexikon der Wissensressource übernommen und können daher bei der semantischen Indexierung auch nicht in den Dokumenten gefunden werden. Nach dem Ausschluss aller ambiger Lexikoneinträge erfüllt die Wissensressource wieder alle geforderten Konsistenzbedingungen.

Ebenso ist es möglich, die Ambiguitäten mit in die Wissensressourcen der semantischen Indexierung mittels der Einführung einer spezieller Relation umzulenken. Der ambige Lexikoneintrag wird dabei einem neu eingeführten Konzept zugeordnet, um die Konsistenzbedingung für die expliziten Wissensressourcen der semantischen Indexierung zu erfüllen.

Als dritte Vorgehensweise können die Ambiguitäten schon beim Aufbau der Wissensressourcen aufgelöst werden. Hierbei werden die ambigen Lexikoneinträge durch Einführen vager Konzepte (siehe unten) disambiguiert. So können die vormals ambigen Lexikoneinträge bei der semantischen Indexierung wie alle anderen eindeutigen Lexikoneinträge behandelt werden.

Die Entfernung aller ambiger Lexikoneinträge aus den Wissensressourcen ist die einfachste der möglichen Vorgehensweisen. Leider gehen auf diese Weise oft auch sinnvolle natürlichsprachliche Ausdrücke verloren, die ansonsten Hinweise auf die thematische Sortierung von Dokumenten gegeben hätten. Da die anderen beiden Vorgehensweisen die ambigen Lexikoneinträge in den expliziten Wissensressourcen erhalten und somit weniger Informationen in den Eingabedaten verloren gehen, kann es durchaus sinnvoll sein, ambige Lexikoneinträge ausführlicher mit einer der beiden anderen Vorgehensweisen zur Behandlung ambiger Lexikoneinträge zu behandeln.

Umlenkung interner Ambiguitäten

Um ambige Lexikoneinträge in den Wissensressourcen der semantischen Indexierung zu erhalten, können die realen Ambiguitäten, die sie ausdrücken, in die Wissensressource übernommen werden. Hierzu wird, ähnlich wie auch bei der Behandlung von Lexikoneinträgen externer Ambiguitäten, ein neues künstliches ambiges Konzept in die Wissensressource eingeführt, welches dann eindeutig von dem ambigen Lexikoneintrag referenziert wird.

Dieses neue ambige Konzept $k^* \in \mathcal{W}_K$ referenziert dann seinerseits wiederum die entsprechenden Konzepte $k_1, \ldots, k_n \in \mathcal{W}_K; n > 1$, die der ambige Lexikoneintrag ursprünglich referenzierte. Das neue ambige Konzept referenziert dann wiederum diese Konzepte über eine spezielle Relation $\mathrm{Amb} \in \mathcal{W}_R$ der expliziten Wissensressource. Dabei gilt $\mathrm{Amb}(k^*, k_i)$ für $1 \leq i \leq n$. Alle anderen, eindeutigen lexikalischen Zuordnungen der Konzepte $k_1, \ldots, k_n$ bleiben erhalten (vgl. Abbildung 7.2).

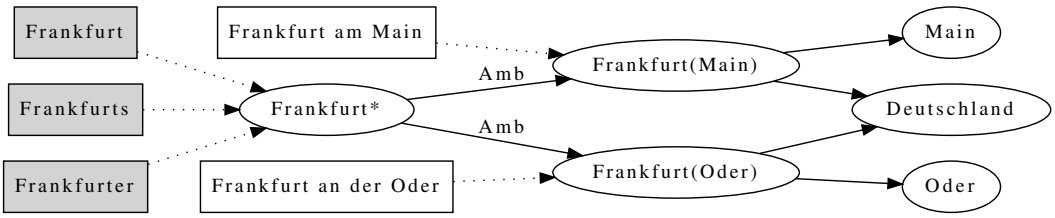


Abbildung 7.2.: Umlenkung interner Ambiguitäten. Die ambigen Begriffe *„Frankfurt"* *„Frankfurter"* und *„Frankfurts"* referenzieren das ambige Konzept *Frankfurt**, welches wiederum über die Relation Amb mit den beiden eindeutigen Konzepten *Frankfurt(Main)* und *Frankfurt(Oder)* verbunden ist. Die eindeutigen Lexikoneinträge referenzieren weiterhin die entsprechenden, eindeutigen Konzepte direkt (vgl. dazu Abbildung 7.3).

Da sie immer über die spezielle Relation Amb mit mindestens einem Konzept verbunden sind, können bei der semantischen Indexierung ambigen Konzepte, egal ob sie durch eine interne oder externe Ambiguität zustande gekommen sind, leicht erkannt werden. Ambige Konzepte interner und externer Ambiguitäten unterscheiden sich in ihrer Repräsentation nur durch die Anzahl ihrer Konzepte, zu denen sie über die spezielle Relation Amb verbunden sind. Aufgrund dieser uniformen Darstellung von internen und externen Ambiguitäten muss bei der semantischen Indexierung nicht weiter zwischen internen und externen Ambiguitäten unterschieden werden.

Wie schon in Abbildung 7.2 zu erkennen ist, können mehrere ambige Lexikoneinträge dasselbe ambige Konzept referenzieren[3] Es muss also nicht für jeden ambigen Lexikoneintrag ein neues Konzept in der Wissensressource erzeugt werden. Ambige Konzepte können in der Wissensressource eindeutig über ihre referenzierten Konzepte identifiziert werden[4]. So muss nicht für jeden ambigen Lexikoneintrag ein eigenes neues Konzept erzeugt werden und die ambigen Konzepte können wie alle anderen Konzepte in den expliziten Wissensressourcen auch von mehreren Lexikoneinträgen gleichzeitig referenziert werden.

Ambige Konzepte externer und interner Ambiguitäten werden immer dann erzeugt und in die Wissensressourcen der semantischen Indexierung eingefügt, wenn Lexikoneinträge mehr als ein Konzept der eigentlichen Wissensressourcen referenzieren. Diese ambigen Konzepte werden dann Teil der expliziten Wissensressourcen der semantischen Indexierung. Dabei gilt es zu beachten, dass ambige

[3]Dies gilt sowohl für interne, als auch für externe Ambiguitäten.

[4]Hierfür kann z.B. ein eindeutiges Benennungsschema verwendet werden, welches ambigen Konzepten auf der Basis der verbundenen Konzepte einen eindeutigen Namen zuordnet.

Konzepte immer nur nicht-ambige Konzepte referenzieren können, da sie nicht Teil der ursprünglichen Wissensressource sind, die zum Aufbau der expliziten Wissensressourcen verwendet werden.

Sie werden immer nachträglich, wenn der Graph der Wissensressource — der ja per Definition nur aus eindeutigen Konzepte besteht — schon erzeugt wurde, aus ein oder mehreren Referenzen auf nicht ambige Konzepte erzeugt. Aus dem gleichen Grund ist es auch ausgeschlossen, dass ein ambiges Konzept von einem nicht ambigen Konzept referenziert wird.

Elimination interner Ambiguitäten durch Einführung vager Konzepte

Beim Umlenken lexikalischer Ambiguitäten durch die Verwendung ambiger Konzepte in der Wissensressource bleiben die Ambiguitäten bei einer semantischen Indexierung erhalten. Sie müssen daher bei der Indexierung und bei Suchanfragen immer gesondert behandelt werden. Dies kann verhindert werden, indem interne Ambiguitäten schon beim Aufbau der Wissensressource eliminiert werden. So müssen Ambiguitäten erst gar nicht in die Wissensressourcen übernommen werden.

Hierzu wird für ambige Lexikoneinträge aus den Eingabedateien, die mehr als ein Konzept $k_1, \ldots, k_n \in \mathcal{W}_K; n > 1$ referenzieren, wiederum ein neues *vages Konzept* $k' \in \mathcal{W}_K$ erzeugt. Dieses neu erzeugte Konzept, wird nun aber nicht mit den an der Ambiguität beteiligten Konzepte $k_1, \ldots, k_n$ verbunden, sondern mit dem Durchschnitt über die Bildmengen aller beteiligter Konzepte $k_1, \ldots, k_n$. Es gilt somit für die neuen Verbindungen von k':

$$\forall R \in \mathcal{W}_R : \forall k \in \bigcap_{k'' \in \{k_1, \ldots, k_n\}} R(\{k''\}) : R(k', k)$$

Im Falle von EFGT-Netzen ist die Intuition einfach. Das neue Konzept k' wird in $R \in \mathcal{W}_R$ mit allen Konzepten im Durchschnitt über die Bildmengen der Konzepte $k_1, \ldots, k_n$ verbunden. Im Falle von Ontologien müssen die Verbindungen für jede Relation in der Ontologie einzeln berechnet werden.

Beispiel 7.2.1. Eine einfache Ontologie enthalte unter anderem die beiden Konzepte *Gerhard Schröder(CDU)* und *Gerhard Schröder(SPD)*. Das Konzept Gerhard Schröder(CDU) sei über die Relation *Beruf* mit dem Konzept *Politiker*, über die Relation *Staatsangehörigkeit* mit dem Konzept *Deutschland* und über die Relation *Partei* mit dem Konzept *CDU* verbunden. Das Konzept *Gerhard Schröder(SPD)* dagegen sei über *Beruf* mit *Politiker*, über *Staatsangehörigkeit* mit *Deutschland* und über *Partei* mit dem Konzept *SPD* verbunden.

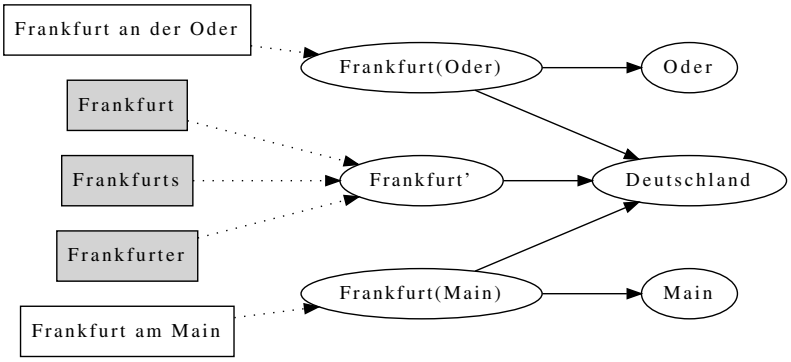


Abbildung 7.3.: Auflösung interner Ambiguitäten. Die ambigen Begriffe *„Frankfurt"* *„Frankfurter"* und *„Frankfurts"* referenzieren das neue eingeführte, vage Konzept *Frankfurt'*, welches wiederum mit dem einzigen Konzept *Deutschland* verbunden ist, welches die ursprünglichen Konzepte gemeinsam referenziert (vgl. dazu Abbildung 7.2).

Soll nun der ambige Lexikoneintrag *„Gerhard Schröder"* durch die Einführung eines vagen Konzepts *Gerhard Schröder'* eliminiert werden, wird das neu eingeführte vage Konzept k' über die Relation *Beruf* mit dem Konzept *Politiker* und über die Relation *Staatsangehörigkeit* mit dem Konzept *Deutschland* verbunden. Da die Parteizugehörigkeit der beiden Konzepte unterschiedlich ist, wird das vage Konzept weder mit dem Konzept *SPD* noch mit dem Konzept *CDU* verbunden.

Aus Sicht der semantischen Indexierung sind diese vagen Konzepte nicht mehr ambig. Sie werden bei der semantischen Indexierung genau wie alle anderen eindeutigen Konzepte behandelt. Durch die Einführung vager Konzepte können interne Ambiguitäten in den Wissensressourcen eliminiert werden, wobei für ambige Begriffe eine minimale thematische Einordnung erhalten bleibt. Die thematische Einordnung vager Konzepte stellt die minimale *sichere* Einordnung ambiger Begriffe dar, die aus der Wissensressource abgeleitet werden kann.

Bei der Erzeugung vager Konzepte können verschiedene Verbindungen der ursprünglichen Konzepte verloren gehen, gerade auch wenn die an der Ambiguität beteiligten Konzepte $k_1, \ldots, k_n$ sehr unterschiedliche thematische Verbindungen aufweisen. In Extremfällen können die neuen Konzepte über gar keine Verbindungen mehr verfügen und es geht die gesamte thematische Einordnung der entsprechenden Lexikoneinträge verloren.

Ebenso wie beim Umlenken von Ambiguitäten auch, kann die Erzeugung redundanter vager Konzepte sehr einfach durch ein einheitliches Benennungsschema leicht verhindert werden (siehe oben). So können auch mehrere Lexikoneinträge, die über dieselben internen Ambiguitäten verfügen auf die neu erzeugten Konzep-

te verweisen (vgl. Abbildung 7.3).

Vergleich der Verfahren

Alle drei Verfahren zur Behandlung interner Ambiguitäten in der Wissensressource der semantischen Indexierung verfügen über verschiedene Vor- und Nachteile. Werden ambige Lexikoneinträge gelöscht, wird die Problematik ambiger Konzepte auf äußerst einfache Weise verhindert. Dabei können aber auch eine ganze Reihe guter Lexikoneinträge verloren gehen.

Werden dagegen die internen Ambiguitäten genau wie die externen Ambiguitäten mit in die Wissensressource übernommen, können die Lexikoneinträge auch weiterhin in den Dokumenten gefunden und indexiert werden. Sie müssen dann aber, wie im Folgenden noch dargestellt werden wird, entweder in einem zusätzlichen Schritt disambiguiert werden oder während der Indexierung und bei Suchanfragen gesondert behandelt werden.

Werden die Ambiguitäten beim Aufbau der Wissensressource zur semantischen Indexierung aufgelöst, bleiben die entsprechenden Lexikoneinträge ebenfalls bestehen und die aufgelösten Ambiguitäten müssen bei der weiteren semantischen Indexierung nicht mehr gesondert beachtet werden. Dabei können allerdings bestimmte thematische Einordnungen der Wissensressource verloren gehen.

Grundsätzlich ist es immer dann besser, die internen Ambiguitäten mit in die Wissensressourcen zu übernehmen, wenn die an der Ambiguität beteiligten Konzepte sich sehr stark thematisch voneinander unterscheiden. In diesen Fällen gehen bei der Auflösung der Ambiguitäten zu viel Informationen verloren. Auch ist eine Disambiguierung, die nur bei ambigen Konzepten durchgeführt wird, sicherer, wenn die beteiligten Konzepte sehr unterschiedliche thematische Einordnungen haben (vgl. Kapitel 7.4).

Weisen die beteiligten Ambiguitäten dagegen eine starke thematische Überlappung auf, sind sie oftmals schwerer zu disambiguieren. Auch bleiben bei einer großen Überlappung der verbundenen Konzepte viele der ursprünglichen thematischen Verbindungen der beteiligten Konzepte auch für das neu erzeugte vage Konzept bestehen. So kann eine bessere thematische Einordnung der Textbelege geschehen. Da eben die neu erzeugten vagen Konzepte nicht ambig sind, müssen sie bei der semantischen Indexierung nicht mehr gesondert behandelt werden.

Alle Formen der Behandlung interner und externer Ambiguitäten können immer auch manuell schon beim Aufbau der Wissensressourcen vorgenommen werden. Zur automatischen Behandlung interner Ambiguitäten bietet die Implementierung der semantischen Indexierung mit expliziten Wissensressourcen verschiedene Möglichkeiten (vgl. Kapitel 8.8).

7.3. Semantische Indexierung von Ambiguitäten

Ambige Konzepte, egal ob sie von internen oder externen Ambiguitäten stammen[5], bilden Mehrdeutigkeiten nach, die aufgrund semantischer Ambiguitäten auf der Zeichenebene entstanden sind. Die grundlegende Idee bei der Handhabung ambiger Konzepte ist dabei die folgende: Ambige Konzepte stellen *indirekte* Konzepte dar, die bei der semantischen Indexierung nicht direkt indexiert werden sollen. Vielmehr sollen die von diesen indirekten Konzepten *referenzierten* Konzepte in den Index geschrieben werden, wobei die resultierenden direkten und indirekten Indexeinträge als ambig markiert werden. Wie auch im Falle anonymer Konzepte, werden bei der semantischen Indexierung niemals Postingfiles für ambige Konzepte angelegt.

Die semantische Indexierung benötigt nur wenige Anpassungen zur Behandlung interner und externer Ambiguitäten. Ohne eine Disambiguierung der Ambiguitäten (siehe Kapitel 7.4) werden ambige Textbelege, genau wie fehlerbehafteten Textbelege auch, in den Index geschrieben. Dazu werden die entsprechenden direkten und indirekten Indexeinträge erzeugt und mit einer speziellen Markierung als ambig markiert. Es werden dabei sowohl die indirekten, als auch die indirekten Indexeinträge eines Textbeleges als ambig gekennzeichnet.

7.3.1. Markierung ambiger Textbelege im semantischen Index

Genau wie auch fehlerbehaftete Textbelege, werden ambige Textbelege im Index markiert. Um nicht die Repräsentation der Indexeinträge erweitern zu müssen, kann das gleiche Feld der Indexeinträge verwendet werden, das auch der Speicherung der Levenshteindistanz dient (vgl. Kapitel 6.4). Hierzu wird ein weiteres Bit der 7 Bits der Levenshteindistanz verwendet, um die Ambiguität von Textbelegen in Indexeinträgen zu markieren. Dies verringert natürlich die maximale Levenshteindistanz, die für fehlerbehaftete Textbelege im Index gespeichert werden kann.

Die maximale Levenshteindistanz, die auf diese Weise in Indexeinträgen gespeichert werden kann, wird so von $2^7 - 1 = 127$ auf $2^6 - 1 = 63$ verringert. Diese maximale Levenshteindistanz von 63 ist für praktische Anwendungen immer noch mehr als ausreichend und so muss weder für einen semantischen Index aus EFGT-Netzen oder aus Ontologien ein weiteres Feld zur Markierung ambiger Textbelege in die Indexeinträgen hinzugefügt werden.

[5] Hierbei sind vage Konzepte, die bei der Eliminierung interner Ambiguitäten neu eingeführt werden ausgenommen. Wie bereits erwähnt, sind diese Konzepte für die semantische Indexierung nicht mehr ambig.

7.3.2. Indexerzeugung mit ambigen Konzepten

Ambige Textbelege können bei der semantischen Indexierung einfach über die ambigen Konzepte, von denen sie referenziert werden, identifiziert werden. Wie auch bei fehlerbehafteten Textbelegen werden nicht die ambigen Konzepte selbst, sondern die von den ambigen Konzepten referenzierten Konzepte ganz normal durch die Erzeugung von direkten und indirekten Indexeinträgen indexiert. Dabei werden alle direkten und indirekten Indexeinträge als ambig markiert. So wird die Ambiguität des Textbeleges an alle Indexeinträge für die verbundenen Konzepte in der Wissensressource weiter vererbt.

Im Falle interner Ambiguitäten, bei denen die verschiedenen Konzepte, die von dem ambigen Konzept referenziert werden, wiederum auf unterschiedliche Konzepte verweisen können, kann es bei der Erzeugung des Index zu mehrfachen, überflüssigen Indexeinträgen kommen, wenn mehrere der referenzierten Konzepte ihrerseits wiederum gleiche Konzepte referenzieren.

Solche Überschneidungen können leicht bei der Indexierung ausgefiltert werden, oder aber auch mit in den Index übernommen werden. Da mehrfache überlappende Textbelege aus den Ergebnissen der Suchanfragen automatisch heraus gefiltert werden, sollten diese überflüssigen Indexeinträge kaum ein Problem darstellen. Einzig wenn mit sehr vielen ambigen Indexeinträgen mit vielen Überlappungen und einem sehr großen Index gerechnet wird, sollte überlegt werden, überflüssige Indexeinträge bei der Indexerzeugung herauszufiltern.

Beispiel 7.3.1. Bei der semantischen Indexierung mit der Ontologie „Erster Weltkrieg“ werden in einem Dokument die beiden ambige Textbelege *„Oder”* und *„Frankfurt”* identifiziert, welche jeweils die ambigen Konzepte *Oder(Fluss)** (vgl. Abbildung 7.1) und *Frankfurt** (vgl. Abbildung 7.2) referenzieren.

Bei der Indexerzeugung werden alle direkten und indirekten Indexeinträge der, von den ambigen Konzepten referenzierten Konzepten *Oder(Fluss)*, *Frankfurt(Main)* und *Frankfurt(Oder)* in den Index geschrieben und als ambig markiert.

Die beiden Konzepte *Frankfurt(Main)* und *Frankfurt(Oder)* sind unter anderem beide über die Relation *Country* mit dem Konzept *Deutschland* verbunden. Ohne eine spezielle Filterung werden nun zwei identische indirekte ambige Indexeinträge in dem Postingfile des Konzepts *Deutschland* für den Textbeleg zu *„Frankfurt”* erzeugt: Einmal für die Verbindung zwischen *Frankfurt(Main)* mit *Deutschland* über die Relation *Country* und ein weiteres Mal für die Verbindung zwischen *Frankfurt(Oder)* und *Deutschland* (ebenfalls über die Relation *Country*).

7.3.3. Tolerante Suchanfragen

In den Ergebnissen einfacher Suchanfragen (vgl. Kapitel 5.5) tauchen niemals ambige Textbelege auf. Um sicher zu stellen, dass nur eindeutige Suchtreffer zurückgegeben werden, müssen ambige Suchtreffer bei der Bearbeitung von Suchanfragen herausgefiltert werden. Dabei ist die Erkennung von ambigen Indexeinträgen trivial. Die prozessierten direkten und indirekten Indexeinträge müssen lediglich auf das Bit untersucht werden, welches ambige Einträge markiert.

Um auch ambige Textbelege in den Ergebnissen von Suchanfragen zuzulassen, können *tolerante Suchanfragen* $?^*$ an den Index gestellt werden. Für solche toleranten Suchanfragen stehen dieselben Möglichkeiten zur Verfügung, wie für einfache Suchanfragen auch. Sie werden auf genau die gleiche Weise abgearbeitet, wobei ambige Textbelege nun *nicht* mehr aus den Ergebnissen heraus gefiltert werden und somit in den Ergebnissen von Suchanfragen mit auftreten.

Beispiel 7.3.2. Eine einfache Suchanfrage $?\,\{$*Frankfurt(Main)*$\}$ an den Index aus Beispiel 7.3.1 gibt keine Ergebnisse zu dem Textbeleg *„Frankfurt"* aus. Erst die tolerante Suchanfrage $?^*\,\{$*Frankfurt(Main)*$\}$ liefert auch den Textbeleg zu *„Oder"* zurück.

Ebenso würde die tolerante relationale Suchanfrage $?^*\,\{$*Country*$\}(\{Deutschland\})$ unter anderem auch Ergebnisse zu den ambigen Textbelegen zu *„Frankfurt"* liefern.

7.3.4. Suchanfragen nach ambigen Konzepten

Eine weitere Problemstellung, die sich bei der Behandlung von Ambiguitäten ergibt, stellt die Behandlung ambiger Konzepte in Suchanfragen dar. Da ambigen Konzepten grundsätzlich keine Postingfiles zugeordnet sind, kann eine Suchanfrage[6] nach einem ambigen Konzept keine Ergebnisse zurückliefern. Dieses Verhalten ist eine direkte Folge der hier dargestellten Behandlung von Ambiguitäten bei der semantischen Indexierung mit expliziten Wissensressourcen.

Aus der Sicht eines Benutzers eines semantischen Index kann es aber durchaus sinnvoll sein, Suchanfragen nach ambigen Konzepten zuzulassen. Hierzu müssen bei Suchanfragen ambige Konzepte gesondert behandeln werden. Im einfachsten Fall kann das ambige Konzept aus der Suchanfrage entfernt werden und durch die eindeutigen Konzepte ersetzt werden, die von dem ambigen Konzept referenziert werden.

Es sind aber auch andere Vorgehen denkbar. So können solche Suchanfragen auf die gleiche Weise auch in tolerante Suchanfragen umgewandelt werden. Auch ist es denkbar den Benutzer auch interaktiv die Ambiguität auflösen zu lassen.

[6] Die genaue Art der Suchanfrage spielt dabei keine Rolle.

Beispiel 7.3.3. Die Suchanfrage $?\,\{\textit{Frankfurt*}\}$ nach dem ambigen Konzept *Frankfurt** kann keine Suchergebnisse zurückliefern, da im semantischen Index kein Postingfile für ambige Konzepte angelegt werden.

Die Suchanfrage kann nun einfach umgewandelt werden, indem das ambige Konzept durch die Konzepte ersetzt wird, die es referenziert:
$?\,\{\textit{Frankfurt(Main)}, \textit{Frankfurt(Oder)}\}$.

7.3.5. Ambige Konzepte der unscharfen Suche

Es kann durchaus vorkommen, dass bei der unscharfen Suche auch ein oder mehrere ambige Konzepte gefunden werden. Solche Fälle müssen bei der semantischen Indexierung auch behandelt werden. Dazu werden die fehlerbehafteten Textbelege ganz normal indexiert, indem die von den anonymen Konzepten referenzierten Konzepte mit einem Fehler in den Index geschrieben werden. Dabei werden etwaige ambige Konzepte mit einem Fehler indexiert, indem die — wiederum von dem ambigen Konzept referenzierten Konzepte — mit einem Fehler und einer *Ambiguitätsmarkierung* in den Index geschrieben werden (vgl. Abbildung 7.4).

In solchen pathologischen Fällen entstehen somit ambige Indexeinträge mit einem Levenshteinabstand. Solche Textbelege werden weder von der fehlertoleranten Suche, bei der ambige Indexeinträge ignoriert werden, noch von einer toleranten Suche, bei der fehlerbehaftete Indexeinträge ignoriert werden, gefunden. Aus diesem Grund kann eine fehlertolerante Suche mit einer Fehlerschranke $k \geq 0$ mit einer tolerante Suche zu $?_k^* Q$ kombiniert werden, um auch fehlerbehaftete, ambige Indexeinträge finden zu können.

Es gilt hierbei noch zu beachten, dass — da anonyme Konzepte *nicht* Teil der Wissensressourcen sind, sondern spontan bei der fehlertoleranten Suche erzeugt werden — der umgekehrte Fall nicht eintreten kann. Es kann somit ein anonymes Konzept vorkommen, das ein oder mehrere ambige Konzepte referenziert, nicht aber ein ambiges Konzept, das ein anonymes Konzept referenziert.

Beispiel 7.3.4. Bei der Indexierung des Textbeleges *„Frankent"*, dem bei der unscharfen Suche sowohl das ambigen Konzept *Frankfurt**, als auch das eindeutige Konzept *Franken* zugeordnet wurde (vgl. Abbildung 7.4), wird nun zum einen das Konzept *Franken* ganz normal mit einem Levenshteinabstand von $l = 1$ indexiert. Andererseits werden auch die beiden Konzepte *Frankfurt(Main)* und *Frankfurt(Oder)* — nicht aber das ambige Konzept *Frankfurt** — als ambige Einträge mit einem Levenshteinabstand von $l = 3$ indexiert.

Eine entsprechende, fehlertolerante Suche $?_1\,(\{\textit{Franken}\})$ liefert somit diesen Textbeleg zurück. Um denselben Textbeleg mit einer Suche nach dem Konzept

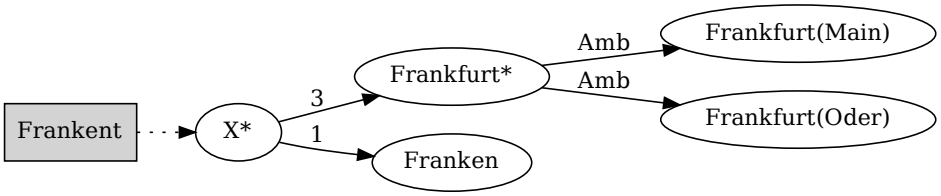


Abbildung 7.4.: Ambige Konzepte bei der unscharfen Suche. Der verstümmelte Textbeleg *„Frankent"* referenziert ein anonymes Konzept X^* der unscharfen Suche. Dieses referenziert nun einerseits das ambige Konzept *Frankfurt** mit einem Levenshteinabstand von 3 und andererseits das (eindeutige) Konzept *Franken* mit einem Levenshteinabstand von 1.

Frankfurt(Main) bzw. *Frankfurt(Oder)* zu finden, muss eine fehlertolerante mit einer toleranten Suche kombiniert werden: $?_3^* (\{\textit{Frankfurt(Oder)}\})$.

7.4. Semantische Auflösung von Ambiguitäten

Ambige Textbelege führen — wie fehlerbehaftete Textbelege auch — bei der semantischen Indexierung zu Mehrdeutigkeiten, die in den Index übernommen werden. Da diese Indexeinträge unter Umständen zu falsch-positiven Suchergebnissen führen können, müssen diese explizit bei der Suche frei geschaltet werden (vgl. Kapitel 7.3.5 und Kapitel 6.6).

Um bei der semantischen Indexierung mit expliziten Wissensressourcen möglichst wenige solcher Mehrdeutigkeiten in den Index aufzunehmen, kann versucht werden, ambige und fehlerbehafteten Textbelege aufzulösen, um mehr eindeutige Textbelege in den Index schreiben zu können. Die sogenannte *Disambiguierung* geschieht nach der exakten und einer optionalen, fehlertoleranten Suche kurz bevor die Textbelege in den Index geschrieben werden (vgl. Abbildung 7.5).

Wie die fehlertolerante Suche auch, stellt die Disambiguierung bei der semantischen Indexierung einen optionalen Schritt dar. Hierzu können verschiedene Disambiguierungsverfahren (siehe unten) hintereinander geschaltet werden um ambige Textbelege aufzulösen. Diese rein optionalen Schritte geschehen nach der strikten und unscharfen Suche bevor die Textbelege in den Index geschrieben werden.

Wenn eine Disambiguierung erfolgreich ist, müssen keine Mehrdeutigkeiten in den Index übernommen werden. Stattdessen werden eindeutige Indexeinträge für

die disambiguierten Konzepte erzeugt.

Durch die Verwendung expliziter Wissensressourcen bei der Indexierung, stehen verschiedene zusätzliche Informationen über die in den Texten gefundenen Konzepte zur Verfügung. Somit müssen die Konzepte nicht isoliert betrachtet werden, sondern können auch in Hinblick auf ihre thematischen Verbindungen untersucht werden. Bettet man die in einem Text vorgefundenen Konzepte in einen Kontext ein, können die verschiedenen thematischen Verbindungen der Konzepte untereinander untersucht werden und für verschiedene Formen der Disambiguierung verwendet werden.

7.4.1. Disambiguierung externer Ambiguitäten

Externe Ambiguitäten können bei der semantischen Indexierung ohne Disambiguierung genau wie interne Ambiguitäten behandelt werden.

Bei der Disambiguierung externer Ambiguitäten muss nun nicht eins von mehreren möglichen Konzepten ausgewählt werden, sondern entschieden werden, ob der Textbeleg tatsächlich das entsprechende Konzept referenziert oder nicht. Da die Disambiguierung auf der Basis der zur Verfügung stehenden expliziten Wissensressourcen geschieht und auch keine weiteren Ressourcen zur Dokumentenanalyse zur Verfügung stehen gestaltet sich die Disambiguierung von externen Ambiguitäten im Allgemeinen schwierig, gerade wenn es sich um die Disambiguierung von Funktionswörtern handelt.

7.4.2. Grundlegende Disambiguierung

Die Disambiguierung muss für anonyme und ambige Konzepte K^*, bevor diese in den Index geschrieben werden, entscheiden, ob eines der referenzierten Konzepte eindeutig in den Index aufgenommen werden kann oder nicht. Dabei werden alle, von ambigen oder anonymen Konzepten referenzierte Konzepte, im Bild $\mathcal{W}_R(K^*)$ (vgl. Definition 1.5.2) von K^* betrachtet. Diese sogenannten *Disambiguierungskandidaten* $K' \in \mathcal{W}_R(K^*)$ stellen die möglichen eindeutigen Konzepte dar, mit denen anonyme Konzepte disambiguiert werden können.

Bei der Überlagerung von anonymen Konzepten K_1^* und ambigen Konzepten K_2^* (vgl. Abbildung 7.4), müssen die Bilder beider Konzepte $\mathcal{W}_R(K_1^*)$ und $\mathcal{W}_R(K_2^*)$ betrachtet werden.

Beispiel 7.4.1. Bei der Disambiguierung des in Abbildung 7.4 dargestellten Textbeleges müssen die *drei* Konzepte *Franken*, *Frankfurt(Main)* und *Frankfurt(Oder)* betrachtet werden. Es muss dann entschieden werden welches der drei möglichen Konzepte in den Index aufgenommen werden soll.

Wenn die Disambiguierung scheitert müssen die entsprechenden Ambiguitäten und Fehler für alle drei Konzepte in den Index übernommen werden (siehe oben).

Die Disambiguierung von K^* ist nicht immer erfolgreich. Sie führt allerdings immer zu einem der folgenden drei Ergebnisse:

1. Genau ein Konzept $K \in \mathcal{W}_R(K^*)$ der von K^* referenzierten Konzepte kann eindeutig für den Textbeleg ausgewählt werden.
2. Keines der von K^* referenzierten Konzepte kann für den Textbeleg ausgewählt werden.
3. Mehr als eins der von K^* referenzierten Konzepte können für den Textbeleg ausgewählt werden.

Im Falle von Punkt 1 war die Disambiguierung erfolgreich. Der entsprechende Textbeleg, der K^* referenziert, kann nun eindeutig dem Konzept $K \in \mathcal{W}_R(K^*)$ zugeordnet werden und normal in den Index eingefügt werden. Alle anderen von K^* referenzierten Konzepte werden verworfen und *nicht* in den Index geschrieben.

Im Falle von Punkt 2 und Punkt 3 dagegen scheitert die Disambiguierung, da entweder kein Konzept eindeutig ausgewählt werden konnte oder da mehrere, unterschiedliche Konzepte Disambiguierungskandidaten für K^* sind. In beiden Fällen wird die Mehrdeutigkeit in den Index übernommen.

7.4.3. Dokumentengedächtnis

Um Ambiguitäten von Textbelegen auflösen zu können, muss der Kontext von Textbelegen im aktuellen Dokument untersucht werden. Der Kontext spiegelt die Themen und Konzepte wieder, die in den vorangegangenen Teilen des Dokuments vor einem Textbeleg gefunden werden konnten. Sowie die semantische Indexierung Dokumente thematisch indexiert, indem nicht nur die gefundenen Konzepte selbst, sondern auch deren Verbindungen in den Wissensressourcen mit indexiert werden, wird der Kontext dazu verwendet, um eine thematische Analyse der Umgebung in den Dokumenten zu ermöglichen.

Aus diesem Grund wird bei der Disambiguierung immer auch ein spezieller *Dokumentenkontext* bzw. ein *Gedächtnis* benötigt. Bei der Disambiguierung wird dabei für jedes Dokument der Kollektion ein lokales Dokumentengedächtnis verwendet, das während der Disambiguierung nach und nach aufgebaut wird. Dazu werden die Konzepte in derselben Reihenfolge, in der sie in den Eingabedokumenten

vorkommen, bearbeitet. Alle eindeutigen, nicht ambigen Konzepte werden dann in das Gedächtnis eingefügt.

Können in einem Dokument nur ambige Konzepte gefunden werden, bleibt das zugehörige Gedächtnis des Dokuments leer. Somit kann keine Disambiguierung durchgefürht werden. Auch aus diesem Grund sollten die expliziten Wissensressourcen immer auch eindeutige Lexikoneinträge enthalten (vgl. auch Kapitel 7.1), da anderenfalls niemals eindeutige Konzepte in den Dokumenten identifiziert werden können und so auch keine Disambiguierung durchgeführt werden kann.

Während der Disambiguierung werden eindeutige oder disambiguierte[7] Konzepte und deren Verbindungen in das Gedächtnis eingefügt. Das lokale Gedächtnis von Dokumenten verfügt über eine fest einstellbare Maximalgröße $N > 0$ um auch bei großen Dokumenten, die verschiedene disjunkte Themengebiete behandeln, einen möglichst lokalen Kontext betrachten zu können. Die Maximalgröße des Gedächtnisses kann natürlich auch so groß gewählt werden, dass immer die gesamte Vergangenheit eines Textbeleges im Dokument betrachtet werden kann.

Am Anfang der Disambiguierung ist das Gedächtnis noch leer und die einzelnen Konzepte werden in der gleichen Reihenfolge, in der sie in dem Dokument identifiziert wurden, nacheinander in das Gedächtnis eingefügt. Solange die Anzahl der in dem Gedächtnis enthaltenen Konzepte noch kleiner oder gleich der Maximalgröße N des Gedächtnisses ist, werden neue Konzepte einfach hinten angefügt. Sobald das Gedächtnis genau N Einträge enthält, überschreiben neu eingefügte Konzepte ältere Konzepte im Gedächtnis. Das Gedächtnis umfasst somit immer nur maximal N Konzepte.

Man kann sich das Dokumentengedächtnis wie eine Uhr als einen Kreis aus einer festen Anzahl von N Feldern mit einem Zeiger in der Mitte vorstellen. Der Zeiger zeigt immer auf das nächste Feld, in das ein neues eindeutiges Konzept eingefügt wird. Wenn ein Konzept in ein Feld eingefügt wird, wird es in das Feld geschrieben, auf das der Zeiger zeigt und der Zeiger wird ein Feld weiter gerückt. Anfänglich sind alle Felder des Kreises noch leer und der Zeiger zeigt auf ein beliebiges leeres Feld. Nach und nach werden einzelne Konzepte in die noch leeren Felder eingetragen. Sobald N Konzepte eingefügt worden sind, steht der Zeiger wieder in seiner Ausgangsposition. Neue Konzepte überschreiben von nun an die Konzepte in den entsprechenden Feldern.

Das Dokumentengedächtnis $\mathcal{M}_d^N$ eines Dokuments d mit einer Maximalgröße von $N > 0$ enthält genau die Konzepte $k \in \mathcal{W}_K$ der expliziten Wissensressource die zu einem Zeitpunkt t während der Disambiguierung in dem Dokument vorgefunden und in das Gedächtnis eingefügt wurden. Das Dokumentengedächtnis

[7] gemeint sind hier ambige Konzepte die im Sinne von Punkt 1 (vgl. Kapitel 7.4) aufgelöst werden konnten.

gibt keinerlei Auskünfte über irgendwelche anderen natürlichsprachlichen Konstrukte in der näheren Umgebung von gefundenen Textbelegen zu Konzepten der Wissensressourcen. Es enthält lediglich die letzten N nicht ambigen Konzepte, die während der Disambiguierung im Dokument vorgefunden wurden.

Um mit einem Dokumentengedächtnis ambige Textbelege zu disambiguieren, muss das Gedächtnis drei Operationen zur Verfügung stellen, mit denen auf den im Gedächtnis abgelegten Dokumentenkontext zugegriffen werden kann:

1. $count(\mathcal{M}_d^N, K)$ zählt die Anzahl der Vorkommen des Konzepts $K \in \mathcal{W}_K$ in $\mathcal{M}_d^N$.

2. $len(\mathcal{M}_d^N)$ gibt die Anzahl der Konzepte im Gedächtnis zurück.

3. $elems(\mathcal{M}_d^N) := \{K \in \mathcal{W}_K | count(\mathcal{M}_d^N, K) > 0\}$ gibt die Menge der *unterschiedlichen* Konzepte in $\mathcal{M}_d^N$ zurück.

Für $len(\mathcal{M}_d^N)$ gilt zu beachten, dass nicht notwendigerweise immer $len(\mathcal{M}_d^N) = N$ gilt, da bei einem noch nicht vollständig gefüllten Gedächtnis am Anfang des Dokuments auch $len(\mathcal{M}_d^N) < N$ gelten kann.

Das Gedächtnis ermöglicht es, die Vorkommen von nicht ambigen Konzepten in der kürzeren Historie des Dokuments abzubilden. Um auch zusätzlich noch die Verbindungen der Konzepte untereinander — die ja das in den Wissensressourcen abgebildete Weltwissen kodieren — berücksichtigen zu können, sollten nicht nur die Konzepte, die direkt im Gedächtnis gespeichert sind, sondern auch die durch diese referenzierten Konzepte beachtet werden. Hierzu werden noch zwei weitere verallgemeinerte Operationen auf dem Dokumentengedächtnis benötigt:

1. $count^*(\mathcal{M}_d^N, K) := \sum_{\forall K' \in \mathcal{W}_K} \begin{cases} count(\mathcal{M}_d^N, K') & \text{falls } K' = K \vee K' \in \mathcal{W}_R(K) \\ 0 & \text{ansonsten} \end{cases}$
   zählt die Anzahl der Vorkommen des Konzepts $K \in \mathcal{W}_K$ in $\mathcal{M}_d^N$ und zusätzlich die Anzahl der Vorkommen des Konzepts K in der Bildmenge eines anderen Konzepts im Gedächtnis $K' \neq K$ unter irgendeiner Relation $R \in \mathcal{W}_R$.

2. $elems^*(\mathcal{M}_d^N) := \{K \in \mathcal{W}_K | count^*(\mathcal{M}_d^N, K) > 0\}$ gibt die Menge der *unterschiedlichen* Konzepte in $\mathcal{M}_d^N$ zurück, wobei nun wiederum auch Konzepte beachtet werden, die sich in der Bildmenge eines Konzeptes im Gedächtnis unter irgendeiner Relation $R \in \mathcal{W}_R$ befinden.

Beide hier dargestellte Operationen funktionieren ähnlich wie die beiden bereits gezeigten Operationen. Im Gegensatz zu diesen betrachten sie aber nicht nur die im Gedächtnis befindlichen Konzepte, sondern auch die von diesen referenzierten

Konzepte. Mit diesen beiden zusätzlichen Operationen auf dem Dokumentengedächtnis können nun verschiedene Formen der Disambiguierung vorgenommen werden.

Beispiel 7.4.2. Eine Disambiguierung eines Dokuments d mit dem Dokumentenkontext $\mathcal{M}_d^N$ mit $N = 2$ startet mit einem leeren Gedächtnis $len(\mathcal{M}_d^N) = 0$. Es gilt somit auch:

- $elems(\mathcal{M}_d^N) = \emptyset$
- $elems^*(\mathcal{M}_d^N) = \emptyset$
- $\forall K \in \mathcal{W}_K : count(\mathcal{M}_d^N, K) = count^*(\mathcal{M}_d^N, K) = 0$

Bei der Disambiguierung wird der erste eindeutige Textbeleg zu dem Konzept *Robert Nivelle* (vgl. Abbildung 4.5) gefunden und in den Kontext eingefügt. Es gilt nun:

- $len(\mathcal{M}_d^N) = 1$
- $count(\mathcal{M}_d^N, \textit{Robert Nivelle}) = 1$
- $elems(\mathcal{M}_d^N) = \{\textit{Robert Nivelle}\}$
- $count^*(\mathcal{M}_d^N, \textit{1924}) = 1$
- $count^*(\mathcal{M}_d^N, \textit{Frankreich}) = 1$
- $elems^*(\mathcal{M}_d^N) = \{\textit{Robert Nivelle}, \textit{1856}, \textit{1924}, \textit{General}, \textit{Military}, \textit{Frankreich}\}$

Bei der weiteren Verarbeitung wird das eindeutige Konzept *Frankreich* gefunden und in das Gedächtnis eingefügt. Dieses Konzept referenziert keine weiteren Konzepte der Wissensressource. Es gilt somit auch:

- $len(\mathcal{M}_d^N) = 2$,
- $count(\mathcal{M}_d^N, \textit{Frankreich}) = 1$
- $count^*(\mathcal{M}_d^N, \textit{Frankreich}) = 2$

Als nächstes wird nun nochmals das Konzept *Frankreich* in das Gedächtnis eingefügt. Da die Maximalgröße des Gedächtnisses erreicht ist, wird der älteste Eintrag im Gedächtnis (das zuerst eingefügte Konzept *Robert Nivelle*) mit dem neuen Konzept *Frankreich* überschrieben. Somit gilt nun:

- $len(\mathcal{M}_d^N) = 2$,
- $count(\mathcal{M}_d^N, \textit{Frankreich}) = count^*(\mathcal{M}_d^N, \textit{Frankreich}) = 2$
- $elems(\mathcal{M}_d^N) = elems^*(\mathcal{M}_d^N) = \{\textit{Frankreich}\}$

7.4.4. Einfache Disambiguierung

Mit dem oben genannten Dokumentengedächtnis kann bereits eine naheliegende und sehr einfache Form der Disambiguierung durchgeführt werden. Hierzu wird für ein ambiges Konzept K^* genau das eindeutige Konzept der Menge $K \in \mathcal{W}_R(K^*)$ ausgewählt, welches am häufigsten im Gedächtnis vorkommt.

Kann genau ein Konzept K mit einer maximalen Vorkommenshäufigkeit ausgewählt werden, ist K das Ergebnis der Disambiguierung. Werden dagegen mehrere Konzepte mit einer maximalen Vorkommenshäufigkeit gefunden, oder ist die maximale Vorkommenshäufigkeit 0, scheitert die Disambiguierung.

Beispiel 7.4.3. Ein Dokument mit dem Inhalt *„Frankfurt am Main ist mit gut 730.000 Einwohnern die größte Stadt Hessens und die fünftgrößte Stadt Deutschlands. Die kreisfreie Stadt ist Zentrum des Ballungsraums Frankfurt mit etwa 2,3 Millionen[2] Einwohnern."*[63] wird mit der Ontologie „Erster Weltkrieg" indexiert (vgl. Abbildung 7.2).

Bei der Indexierung wird unter anderem auch der Textbeleg *„Frankfurt"*[8], der mit dem ambigen Konzept *Frankfurt** verbunden ist, gefunden (vgl. Abbildung 7.2). Zur Disambiguierung dieses Textbeleges, werden nun die Vorkommenshäufigkeiten der beiden Konzepte *Frankfurt(Main)* und *Frankfurt(Oder)* im Dokumenten Gedächtnis berechnet[9].

Da wegen des Textbeleges *„Frankfurt am Main"*[10] $count^*(\mathcal{M}_d^N, \textit{Frankfurt(Main)}) = 1$ gilt und dagegen $count^*(\mathcal{M}_d^N, \textit{Frankfurt(Oder)}) = 0$ gilt, kann der Textbeleg dem eindeutigen Konzept *Frankfurt(Main)* zugeordnet werden. Das ambige Konzept *Frankfurt** kann somit als das eindeutige Konzept *Frankfurt(Main)* disambiguiert werden. Es wird somit als eindeutiger Textbeleg in den Index aufgenommen.

Die einfache Disambiguierung funktioniert besonders gut, wenn einerseits alle möglichen Kandidaten für eine Disambiguierung durch eindeutige Lexikoneinträge referenziert werden und andererseits diese eindeutigen Lexikoneinträge auch im Dokument auftreten. Eindeutige Lexikoneinträge helfen hier bei der Disambiguierung. Dabei kann das Phänomen ausgenützt werden, dass in natürlichsprachlichen Texten oftmals ambige Ausdrücke erst dann verwendet werden, wenn sie vorher durch eindeutige Ausdrücke für den Leser konkretisiert wurden (vgl. Beispiel 7.4.3).

Da bei der einfachen Disambiguierung nur absolute Vorkommenshäufigkeiten verglichen werden und auch nur die zu disambiguierende Konzepte selbst, nicht

[8] *„[...] Ballungsraum Frankfurt mit [...]"*

[9] Das Dokumentengedächtnis in diesem Beispiel sei groß genug um alle bisher im Dokument angetroffenen Konzepte vorzuhalten.

[10] *"Frankfurt am Main ist mit [...]"*

aber deren thematische Einordnungen, zur Disambiguierung herangezogen werden, kann dieses Verfahren scheitern. Gerade wenn die zu disambiguierenden Konzepte nicht direkt im Dokument als eindeutige Textbelege, sondern durch ihre thematische Einordnung implizit auftauchen und auch nicht durch andere Konzepte im Dokumentengedächtnis referenziert werden, fehlen die entsprechenden Vorkommen von Kandidaten zur Disambiguierung und es kann keine Disambiguierung der ambigen Konzepte durchgeführt werden.

7.4.5. Regelbasierte Disambiguierung

Regelbasierte Disambiguierung wendet konzeptabhängige, manuell erstellte *Disambiguierungsregeln* an, um Textbelege zu disambiguieren. Diese Regeln ermöglichen es Zusammenhänge zwischen Konzepten zu formulieren, die dabei helfen sollen Ambiguitäten aufzulösen.

Disambiguierungsregeln verwenden das Dokumentengedächtnis zur kontextabhängigen Berechnung beliebiger Zusammenhänge zwischen den Konzepten der Wissensressource. Hierzu wird jedem Konzept $K \in \mathcal{W}_K$ eine Disambiguierungsregel $P_K : \mathcal{M}_d^N \times \mathcal{W} \rightarrow \{\textit{wahr}, \textit{falsch}\}$ zugeordnet, wobei jedem Konzept, welches über keine manuell erstellte Regel verfügt, implizit die *leere Disambiguierungsregel* $P_\varepsilon = \textit{falsch}$ zugeordnet wird.

Um ein ambiges Konzept K^* zu disambiguieren, wird für jedes von dem ambige Konzept referenzierte Konzept $K' \in \mathcal{W}_R(K^*)$ das Prädikat $P_{K'}$ berechnet. Wenn genau ein Prädikat P_K *wahr* zurückliefert, wird K^* als K disambiguiert. Andernfalls scheitert die Disambiguierung von K^*.

Beispiel 7.4.4. Ein einfaches Prädikat eines Konzepts K $P_K(\mathcal{M}_d^N) = count^*(\mathcal{M}_d^N, A) > count^*(\mathcal{M}_d^N, B)$ liefert genau dann *wahr* zurück, wenn das Konzept A häufiger im aktuellen Dokumentenkontext $\mathcal{M}_d^N$ vorkommt als das Konzept B.

Die einfache Disambiguierung des ambigen Konzepts *Frankfurt** aus Beispiel 7.4.3 kann somit nachgebildet werden, indem den beiden Konzepten *Frankfurt(Main)* und *Frankfurt(Oder)* folgende Disambiguierungsregeln zugeordnet werden:

- $P_{\textit{Frankfurt(Main)}} = count^*(\mathcal{M}_d^N, \textit{Frankfurt(Main)}) > count^*(\mathcal{M}_d^N, \textit{Frankfurt(Oder)})$
- $P_{\textit{Frankfurt(Oder)}} = count^*(\mathcal{M}_d^N, \textit{Frankfurt(Oder)}) > count^*(\mathcal{M}_d^N, \textit{Frankfurt(Main)})$

Bei der Disambiguierung des Konzepts *Frankfurt** werden dann die Ergebnisse der beiden Disambiguierungsregeln $P_{\textit{Frankfurt(Main)}}$ und $P_{\textit{Frankfurt(Oder)}}$ berechnet. Genau dann, wenn eine der beiden Disambiguierungsregeln *wahr* zurückliefert, kann *Frankfurt** als das entsprechende, zugrundeliegende Konzept disambiguiert werden.

Mit Disambiguierungsregeln können gezielt Bedingungen formuliert werden, die bei der Disambiguierung verwendet werden sollen. Dabei müssen nicht für alle Konzepte der Wissensressource Regeln erstellt werden. Vielmehr sollten nur Konzepte der Wissensressource, die bekannte, pathologische Mehrdeutigkeiten aufweisen, mit solchen Regeln versehen werden.

Die Disambiguierungsregeln können bei der Berechnung alle Konzepte der Wissensressource verwenden. So können beliebige Bedingungen und Zusammenhänge zur Disambiguierung von Konzepten in den expliziten Wissensressourcen formuliert werden. Je nach der genauen Implementierung der Regeln können auch sehr komplexe Berechnungen durchgeführt werden.

Gerade auch für Ontologien, bei denen die Konzepte unter einer Vielzahl verschiedener Relationen mit anderen Konzepten der Wissensressource verbunden sind, können die Disambiguierungsregeln konkrete Hinweise zur Disambiguierung liefern. Sie können dazu verwendet werden, den Dokumentenkontext auf signifikante Vorkommenshäufigkeiten bestimmter konzeptabhängiger Schlüsselkonzepte zu untersuchen und im gegebenen Fall gezielt eine Ambiguitätsauflösung zu ermöglichen oder auch zu unterbinden.

Beispiel 7.4.5. Mit den Prädikaten der regelbasierten Disambiguierung können auch komplexe Berechnungen auf dem Dokumentenkontext durchgeführt werden. Das Prädikat

$$P_K(\mathcal{M}_d^N) = \frac{count^*(\mathcal{M}_d^N, A)}{\max\{count^*(\mathcal{M}_d^N, B) : B \in elems^*(\mathcal{M}_d^N)\}} > w_K$$

liefert genau dann wahr zurück, wenn die normalisierte Dichte der Suchwörter des Konzepts A größer als irgendein (konzeptabhängiger) Schwellenwert w_K ist.

Bei der regelbasierten Disambiguierung muss auf die Struktur der zugrundeliegenden Relationen geachtet werden. Im Falle von EFGT-Netzen, bei denen ja jedes Konzept mit dem Wurzelknoten *Topnode* verbunden ist, würde die Formel im Zähler als Maximum immer die Anzahl *aller* Konzepte im Kontext zurückgeben. Für EFGT-Netze müsste die oben genannte Formel folgendermaßen angepasst werden:

$$P_K(\mathcal{M}_d^N) = \frac{count^*(\mathcal{M}_d^N, A)}{\max\{count^*(\mathcal{M}_d^N, B) : B \in (elems^*(\mathcal{M}_d^N) \setminus \{\textit{Topnode}\})\}} > w_K$$

Die Disambiguierungsregeln werden beim Aufbau der Wissensressourcen der semantischen Indexierung eingelesen und müssen vor der Verwendung analysiert werden. Die Regeln stellen eine eigene domänenspezifische Sprache dar, die von der Implementierung der semantischen Indexierung eingelesen und in eine ausführbare Form gebracht werden muss. Die vorliegende Implementierung von `semix`

implementiert eine eigene domänenspezifische Sprache zur Formulierung solcher Disambiguierungsregeln (vgl. Kapitel B.4), die zur regelbasierten Disambiguierung eingesetzt werden kann.

Die regelbasierte Disambiguierung erlaubt eine flexible Disambiguierung von Konzepten. Dabei können alle möglichen Konzepte der Wissensressource — auch solche die in der Wissensressource gar nicht mit den beteiligten Konzepten verbunden sind — für die Disambiguierung verwendet werden. Bei der regelbasierten Disambiguierung müssen nicht für alle Konzepte Regeln erzeugt werden. So können gezielt für einige wenige problematische Ambiguitäten in den Wissensressourcen spezielle Disambiguierungsregeln formuliert werden um die Qualität der semantischen Indexierung zu verbessern.

Da jedoch jede Regel von Hand erstellt werden muss, ist ihre Verwendung mit einem großen manuellen Aufwand verbunden. Auch kann es oftmals schwierig sein, stabile Regeln zur Disambiguierung zu formulieren.

7.4.6. Thematische Disambiguierung

Während die einfache Disambiguierung die genauen konzeptuellen Verbindungen der zu disambiguierende Konzepte außer Acht lässt und die regelbasierte Disambiguierung mit einem großen manuellen Aufwand verbunden ist, wird bei der *thematischen Disambiguierung* versucht, die Verbindungen der Konzepte in der Wissensressource selbst, ohne zusätzlichen manuellen Aufwand zur Disambiguierung auszunutzen. Hierzu werden die Verbindungen der Konzepte, wie sie in den Wissensressourcen abgebildet sind, direkt zur Disambiguierung von ambigen Konzepten herangezogen. Wie bisher auch, werden dabei die Relationen, unter denen die Konzepte miteinander verbunden sind, ignoriert und nur die Bildmenge $\mathcal{W}_R(K^*)$ ambiger Konzepte K^* betrachtet.

Um ein ambiges Konzept $K^* \in \mathcal{W}_K$ zu disambiguieren, wird für jeden Disambiguierungskandidaten $K' \in \mathcal{W}_R(K^*)$ die Bildmenge aller mit K' verbundenen Konzepte $\mathcal{W}_R(K')$ berechnet. Diese Menge wird nun mit der Menge aller Konzepte im aktuellen Dokumentengedächtnis verglichen und die *thematische Überlappung*

$$o_{K'} := \frac{|\mathcal{W}_R(K') \cap elems^*(\mathcal{M}_d^N)|}{|elems^*(\mathcal{M}_d^N)|}$$

der Bildmenge $\mathcal{W}_R(K')$ mit allen Konzepten im Gedächtnis $elems^*(\mathcal{M}_d^N)$ berechnet.

Wie auch schon bei der einfachen Disambiguierung wird das Konzept K' mit einer eindeutigen maximalen Überlappung $o_{K'} > w_o$ ausgewählt, welches eine größte thematische Überlappung mit den Themen im Dokumentengedächtnis auf-

weist. Existiert kein solchen Konzept oder falls mehrere solche Konzepte existieren, scheitert die thematische Disambiguierung und die Ambiguität kann nicht eindeutig aufgelöst werden. Der globale *Schwellwert* $0 \leq w_o \leq 1$ gibt dabei den minimalen Grad der Überlappung an, ab dem ein möglicher Kandidat als Disambiguierung akzeptiert wird.

Die thematische Disambiguierung berechnet für jedes der eindeutigen Konzepte die prozentuale Überschneidung mit allen Konzepten im Gedächtnis. Konzepte, deren verbundene Konzepte allesamt in $elems^*(\mathcal{M}_d^N)$ enthalten sind, haben eine Überlappung von $o_{K'} = 1$, Konzepte mit keinerlei Überschneidungen dagegen eine Überlappung von $o_{K'} = 0$.

Die thematische Disambiguierung verwendet die thematische Einordnung der Konzepte im Gedächtnis, um zu messen, wie gut die einzelnen Disambiguierungskandidaten zu den vorhandenen Themen im Gedächtnis passen. So können Ambiguitäten zu thematisch sehr unterschiedlichen Konzepten automatisch aufgelöst werden, indem der thematische Dokumentenkontext untersucht wird.

Für thematisch ähnliche Konzepte funktioniert dieses Verfahren zur Ambiguitätsauflösung nicht sehr stabil. Solche Fälle können entweder mit der einfachen Disambiguierung oder der regelbasierten Disambiguierung behandelt werden. Auch sollte bei solchen Fällen immer überlegt werden, ob es nicht günstiger ist, die Ambiguitäten durch das Einführen vager Konzepte in die Wissensressource (vgl. Kapitel 7.2.2) auf der Ebene der expliziten Wissensressourcen aufzulösen.

7.5. Zusammenfassung

In diesem Kapitel wurde nun abschließend die Möglichkeiten der Behandlung von Ambiguitäten bei der semantischen Indexierung mit expliziten Wissensressourcen dargestellt. Während die Konzepte einer Wissensressource immer eindeutig sind, sind dies die natürlichsprachlichen Ausdrücke, welche die Konzepte referenzieren, oftmals nicht. Gerade Namen von Personen und Organisationen und auch geologischen Entitäten wie Städte, Flüsse und Gebirge sind dabei meist hochgradig ambig. Auch können durch die fehlertolerante Suche weitere Ambiguitäten hinzukommen.

Bei der semantischen Indexierung werden Ambiguitäten — sofern sie nicht schon beim Aufbau der Wissensressourcen aufgelöst wurden — durch spezielle Konzepte abgebildet, die von den ambigen Lexikoneinträgen referenziert werden und ihrerseits mögliche Disambiguierungskandidaten referenzieren. Bei der Indexierung werden dann nicht die ambigen Konzepte selbst, sondern deren referenzierte, eindeutige Konzepte indexiert und im Index speziell ausgezeichnet. So können diese Einträge für ambige Textbelege bei der Suche speziell behandelt werden.

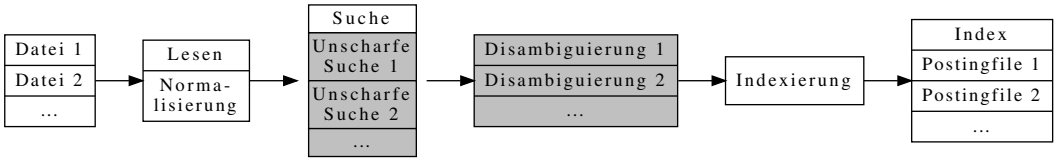


Abbildung 7.5.: Die möglichen Verarbeitungsschritte bei der semantischen Indexierung mit expliziten Wissensressourcen. Optionale Verarbeitungsschritte sind dabei grau hinterlegt.

Wie dargestellt wurde, kann die fehlertolerante Suche nicht nur zu einfachen Mehrdeutigkeiten führen, sondern sogar noch zu komplexeren, wenn Ergebnisse der unscharfen Suche ambige Konzepte referenzieren. In solchen Fällen müssen sowohl die Fehler als auch die Ambiguitätsmarkierungen in den Index übernommen werden.

Um die Anzahl ambiger Indexeinträge zu vermindern und so die Qualität des semantischen Index zu vergrößern, können ambige Textbelege disambiguiert werden. Dabei wird versucht über einen Dokumentenkontext ein Konzept der möglichen Disambiguierungskandidaten eindeutig auszuwählen und zu indexieren. Es gilt aber immer zu beachten, dass gerade auch Namen von Personen nur sehr schwer richtig disambiguiert werden können. Es ist daher oftmals die einfachere und pragmatischere Lösung, die Ambiguitäten in den Index zu übernehmen und nur für Suchergebnisse zuzulassen, wenn sie dies explizit fordern oder automatisch, wenn zu wenig Suchergebnisse für Anfragen produziert werden können.

Ebenso wie bei der fehlertoleranten Suche, bei der mehrere Suchdurchläufe mit ansteigenden maximalen Fehlerschranken k hintereinander geschaltet werden können, können bei der Disambiguierung mehrere Verfahren miteinander verkettet werden. Dabei verwaltet jedes Disambiguierungsverfahren sein eigenes lokales Dokumentengedächtnis und es wird nur versucht Ambiguitäten zu disambiguieren, die nicht von einem vorgelagerten Verfahren disambiguiert werden konnten.

Abbildung 7.5 stellt nun abschließend alle möglichen Verarbeitungsschritte bei der semantischen Indexierung mit expliziten Wissensressourcen dar. Die Disambiguierung wird dabei immer zwischen die Suche und die abschließende Indexierung geschaltet werden.

8. Semix

Die in dieser Arbeit dargestellte semantische Indexierung mit expliziten Wissensressourcen wurde parallel zu dieser Arbeit prototypisch umgesetzt. Das Programm Semix *baut eine Wissensressource aus einer Reihe von Eingabedateien auf, erzeugt mit Hilfe der Wissensressource einen semantischen Index aus einer Menge von Dokumenten und ermöglicht Anfragen an den resultierenden Index. Genauso sind in dem Programm auch die fehlertolerante Suche und die Behandlung von Ambiguitäten umgesetzt. In diesem Kapitel sollen nun verschiedene Aspekte der Implementierung einer semantischen Indexierung am Beispiel von Semix dargestellt werden.*

8.1. Semix

Semix ist als Client- und Serveranwendung realisiert. Ein Server hält dabei alle internen Datenstrukturen der Wissensressource vor und bietet eine einfache Schnittstelle zur Suche auf der Wissensressource, zur Dokumentenindexierung und zur Indexsuche an. Verschiedene Clients können einfache Anfragen an diesen Server stellen, um Dokumente zu indexieren oder Suchen auf dem Index und der Wissensressource auszuführen. Diese Architektur hat unter anderem den Vorteil, dass die Wissensressourcen nicht bei jedem Programmaufruf neu erzeugt bzw. geladen werden müssen, sondern permanent vorgehalten werden können.

Semix umfasst neben dem Server sowohl einen einfachen, kommandozeilenorientierten Client, als auch einen einfachen HTTP-Server, mit dem auch webbasierte Anfragen an den Server gestellt werden können. Die Anwendung und deren Quellcode stehen zur freien Verfügung. Näheres zur Installation der Software findet sich in Appendix A. In Appendix B ist die genaue Verwendung von Semix dokumentiert.

8.2. Interne Repräsentation der expliziten Wissensressourcen

Jedes Konzept der Eingabedateien muss in Semix immer eindeutig über eine URI identifiziert sein. Neben der eindeutigen URI verfügt jedes Konzept über eine eindeutige interne Identifikationsnummer $k_{id} > 0$, die intern als Index verwendet wird, um Konzepte effizient in konstanter Zeit nachzuschlagen zu können.

Semix verwaltet nicht nur die Konzepte der Wissensressource, sondern auch deren Prädikate. Die Prädikate werden dabei genau wie Konzepte behandelt — sie dienen später zur Realisierung der Relationen in den expliziten Wissensressourcen. Die Prädikate verfügen ebenfalls über eine eindeutige URI und auch über eine eindeutige Identifikationsnummer. Dieses Vorgehen ist weitestgehend der Implementierung von librdf [7], einer quelloffenen Programmbiliothek für verschiedenen Semantic-WEB [43] Anwendungen, nachempfunden.

Semix unterstützt RDF-XML[43, 60, 61, 62] formatierte Wissensressourcen, welche zur Verarbeitung des TopicZoom-Netzes (vgl. Kapitel 4.3.2) verwendet werden. Darüber hinaus wird auch das Format TURTLE [4] unterstützt, mit dem die Ontologie „Erster Weltkrieg“ (vgl. Kapitel 4.5.2) verarbeitet wird.

Die Wissensressource wird intern als gerichteter, gelabelter Graph repräsentiert. Jeder *Knoten* des Graph repräsentiert dabei entweder ein Konzept oder eine Relation in der Wissensressource. Alle Knoten des Graphen verfügen ihrerseits über eine Menge von *Kanten*. Kanten definieren dabei immer eine gelabelte Verbindung von einem Ursprungsknoten zu einem Zielknoten. Sie sind intern als Paare aus zwei Knoten des Graphen realisiert. Das erste Element einer Kante definiert dabei die Relation, unter der die Konzepte miteinander verbunden sind.

Bei der internen Repräsentation der Wissensressource wird nicht weiter zwischen Prädikat- und Konzeptknoten unterschieden. Konzepte und Prädikate können beliebig miteinander kombiniert werden. Semix schränkt die Verwendung von Prädikaten und Konzepten in keiner Weise ein und ermöglicht so die flexible Verwendung unterschiedlicher Wissensressourcen.

Ebenso unterscheidet Semix in seiner internen Repräsentation nicht zwischen Ontologien oder EFGT-Netzen. Jede Verbindung im Graphen muss neben dem Zielknoten auch das Prädikat der Verbindung auszeichen. Erst bei der Speicherung der Indexeinträge in den Postingfiles werden die Prädikate von EFGT-Netzen und Ontologien unterschiedlich behandelt, indem die entsprechenden Prädikate von indirekten Indexeinträgen entweder mit in den Index geschrieben werden oder nicht.

Für das Lexikon der Wissensressource wird eine eingebaute Hash-Tabelle verwendet, welche die URIs der Konzepte auf eine Liste von Lexikoneinträgen abbildet. Der $\mathcal{W}$-Automat wird aus diesem einfachen Lexikon aufgebaut (siehe un-

ten). Er ist als minimierter Automat in einer verschränkten Übergangstabelle mit konzeptuellen Zuordnungen realisiert. Die Identifikationsnummern der Konzepte dienen dabei der Verbindung zwischen den Lexikoneinträgen des Automaten und den Konzepten des Graphen.

Da Semix nicht nur strikte, sondern auch fehlertolerante Suchen erlaubt, werden beim Aufbau des Lexikonautomaten, wie in Kapitel 6.3.2 dargestellt wurde, die eingeführten Pointer in den Zellen des Lexikonautomaten mit berechnet. So kann der von Semix verwendete $\mathcal{W}_K$-Automat sowohl zur strikten, als auch zur unscharfen Suche eingesetzt werden.

Die Implementierung des Automaten mittels verschränkter Übergangstabellen in Semix verwendet 64 Bit große Zellen, in denen maximal 32 Bit breite Identifikationsnummern gespeichert werden können. Dabei arbeitet Semix ausschließlich auf UTF-8 [69] kodierten Texten. So können die Zeichen und deren Pointer, die in den Übergangszellen gespeichert werden müssen, auf 8 Bit begrenzt werden, da im Falle von Multibyte-Kodierungen mehrere byte-weise Übergänge kodiert werden müssen (siehe dazu auch [19, S.53ff]). Damit ist es möglich, die Bandbreite des gesamten Unicode-Zeichensatzes abzubilden, ohne ein internes Alphabet verwalten zu müssen.

Für Sprachen mit einem großen Anteil lateinischer Buchstaben, welche in UTF-8 mit einem Byte kodiert werden und nur einem geringen Prozentsatz sprachspezifischer Sonderzeichen, die mit UTF-8 als zwei, drei oder vier Bytes kodiert werden, stellt diese Vereinfachung kaum ein Problem dar. Auch handelt es sich bei UTF-8 um ein weit verbreitetes Kodierungsformat [12]. So liegen viele vorhandene Dokumente bereits in dieser Kodierung vor und es müssen oftmals keine zusätzlichen Konvertierungsschritte vorgenommen werden. Auch können mit UTF-8 kodierte Dokumente keine Probleme der sogenannten *endianess*[1] bei der Übertragung von Binärdaten auftreten. Dies vereinfacht auch die Verwendung binärer Serailisierungsformate, da keine Rücksicht auf die Byte-Reihenfolge der Rechner und auch der Übertragungsschnittstellen genommen werden muss.

Die vorliegende Implementierung würde aber grundsätzlich eine Buchstabenbreite von bis zu 16 Bit erlauben, ohne dass die Größe der Zellen erhöht werden müsste. Je nach erwarteten Sprachen müssten dann entsprechende Konvertierungen von und nach UTF-16 [31] vorgenommen werden oder ein maßgeschneidertes internes Alphabet mitsamt entsprechender Konvertierungen verwendet werden. Hierbei sei noch anzumerken, dass auch bei UTF-16 — im Gegensatz zu UCS-2 [33] — Multibyte-Kodierungen in Form sogenannter *surrogate pairs* vorkommen, diese aber selten sind und vor allem Zeichen aus höheren Unicode-Ebenen betreffen.

[1] Deutsch: Byte-Reihenfolge.

8.3. Erzeugung der expliziten Wissensressource

Die explizite Wissensressource wird aus Dateien, die entweder in RDF-XML oder in TURTLE formatiert sind, aufgebaut. Beide Formate folgen einem tripel-basierten Ansatz, in dem die Wissensressource als eine Menge von Tripeln aus Subjekten, Prädikaten und Objekten repräsentiert ist. Eine Konfigurationsdatei erlaubt es dabei, den genauen Aufbau der expliziten Wissensressourcen aus den Eingabedateien zu steuern.

Die Unterstützung von RDF-XML formatierten Dateien ermöglicht die direkte Verwendung der TopicZoom-Netze, deren Eingabedateien in diesem Format vorliegen. Die Eingabedateien der Ontologie „Erster Weltkrieg", die wiederum ihr eigenes Format verwendet, müssen dagegen mit einer Reihe von Skripten nach entweder TURTLE oder RDF-XML konvertiert werden, dabei werden auch verschiedene Erweiterungen in die Ontologie eingefügt. So werden verschiedene unter anderem auch natürlichsprachliche Lexikoneinträge für Datumsangaben in verschiedenen Sprachen erzeugt und die implizite Typisierung der Ontologie explizit aufgelöst, indem Konzepte für die entsprechenden Typen erzeugt werden.

Aus den Eingabedateien werden die enthalten Tripel ausgelesen und aus diesen dann die interne Repräsentation der Wissensressource aufgebaut. Der Parser gibt eine Reihe von Subjekt-, Prädikat und Objekttripeln aus, wobei jedes Element der Tripel durch eine URI eindeutig identifiziert werden kann. Diese URIs werden auch von Semix intern zur eindeutigen Identifikation der Elemente weiter verwendet. Die unterschiedlichen Prädikate der Objekttripel dienen dabei sowohl zur Auszeichnung verschiedener Verbindungen in unterschiedlichen Relationen als auch zur Verbindung von Lexikoneinträgen mit den Konzepten der Wissensressource.

8.3.1. Aufbau des Graphen

Semix baut aus den Tripeln der Eingabedateien die gesamte interne Repräsentation der Wissensressource auf. Die Konfiguration von Semix (vgl. Appendix B.2) erlaubt es, verschiedene Eigenschaften von Prädikaten zu definieren. So können spezielle *Namensprädikate* definiert werden, die eindeutige oder ambige Vorzugs- und Alternativnamen von Konzepten spezifizieren und der Definition von Lexikoneinträgen für Konzepte dienen. Solche Prädikate werden *nicht* als Teil der Relationsmenge $\mathcal{W}_R$ der expliziten Wissensressource behandelt, sondern dienen nur der Steuerung des Aufbaus der internen Repräsentation der Wissensressource.

Ebenso können relationale Eigenschaften wie Symmetrie oder Transitivität von Prädikaten in den expliziten Wissensressourcen festgelegt werden. Verschiedene Prädikate können auch explizit vom Aufbau der Wissensressource ausgeschlossen

werden. Näheres zur Konfiguration der Wissensressource findet sich in Appendix B.2.

Wie bereits erwähnt wurde, unterscheidet die interne Repräsentation der Wissensressource in Semix nicht zwischen EFGT-Netzen und Ontologien und behandelt diese uniform. Die Unterschiede zwischen Ontologien und EFGT-Netzen können direkt über die Konfiguration des Index (vgl. Appendix A.3), die verwendeten Prädikate und deren Eigenschaften modelliert werden.

Der Graph der internen Repräsentation der Wissensressource wird aus den Tripeln der Eingabedateien aufgebaut, indem alle Tripel in den Eingabedateien, die nicht explizit von der Verarbeitung ausgeschlossen sind, ausgelesen werden und nach ihren Prädikaten sortiert werden. Prädikate, die Lexikoneinträge definieren, werden nicht zum Aufbau des Graphen verwendet. Sie werden erst nachdem der Graph mit allen Knoten und Kanten aufgebaut wurde, verwendet, um das Lexikon der Wissensressource zu erzeugen.

Nachdem alle Tripel der Eingabedateien verarbeitet und nach Prädikaten sortiert worden sind, müssen gegebenenfalls noch verschiedene Anpassungen an den Relationen vorgenommen werden. Je nach Konfiguration können Relationen invertiert werden und symmetrische und transitive Hüllen der Relationen berechnet werden. Neben der trivialen Implementierung zur Berechnung der Invertierung von Relationen und derer symmetrischer Hüllen, verwendet Semix den in [39] beschriebenen Algorithmus zur Berechnung transitiver Hüllen.

Erst nachdem alle Anpassungen und Hüllenbildungen der Relationen berechnet worden sind, werden aus allen Tripeln die entsprechenden Knoten mit ihren Kanten erzeugt. Die URIs der Tripel werden dabei zur eindeutigen Zuordnung von Konzepten und Prädikaten verwendet.

8.3.2. Erzeugung des Lexikonautomaten

Nachdem der Graph mit allen Konzepten erzeugt wurde, wird aus denjenigen Tripeln, die Lexikoneinträge repräsentieren, der Lexikonautomat der Wissensressource erzeugt. Hierzu werden die Lexikoneinträge zuerst normalisiert. Dann werden externe und interne Ambiguitäten behandelt. Für Lexikoneinträge, die über die Konfiguration der Wissensressource als externe Ambiguitäten markiert sind, wird ein neues ambiges Konzept erzeugt, das über eine spezielle Relation $\mathrm{Amb} \in \mathcal{W}_R$ mit dem von dem Lexikoneintrag referenzierten Konzept verbunden wird (vgl. Kapitel 7.2.1) und schließlich dem ambigen Lexikoneintrag zugeordnet. Lexikoneinträge interner Ambiguitäten werden je nach verwendeter Konfiguration behandelt, indem diese entweder verworfen oder entsprechende ambige oder eindeutige Konzepte mit den entsprechenden Verbindungen im Graphen erzeugt werden (vgl. Kapitel 7.2.2), die dann den entsprechenden ambigen Lexikoneinträgen zugeordnet

werden.

Das Vorgehen zur Auflösung interner Ambiguitäten in der Wissensressource kann dabei entweder fest eingestellt werden oder über eine einfache Heuristik dynamisch ausgewählt werden. Hierzu kann ein fester Schwellenwert $0 <= t <= 1$ definiert werden, der den prozentualen Grad der Überschneidung der verbundenen Konzepte angibt, ab dem eine interne Ambiguität aufgelöst wird, indem ein vages Konzept eingeführt wird. Für interne Ambiguitäten wird dann die prozentuale Überlappung der Themen der Konzepte, die an der Ambiguität beteiligt sind berechnet, wobei die Relationen ignoriert werden. Übersteigt dieser Grad den angegebenen Schwellenwert, wird ein vages Konzept für die interne Ambiguität erzeugt und der Lexikoneintrag mit diesem assoziiert. Liegt die prozentuale Überschneidung dagegen unter diesem Schwellenwert, wird ein neues ambiges Konzept eingeführt, mit dem der entsprechende Lexikoneintrag verbunden wird.

Auf diese Weise können interne Ambiguitäten zu thematisch ähnlichen Begriffen durch vage Konzepte und interne Ambiguitäten zu thematisch unterschiedlichen Konzepten durch ambige Konzepte automatisch aufgelöst werden. Dies kann bei der späteren Disambiguierung helfen, die vor allem gut zwischen thematisch unterschiedlichen Konzepten unterscheiden kann.

Beispiel 8.3.1. Der ambige Lexikoneintrag *„Gerhard Schröder"*, der den beiden Konzepten *Gerhard Schröder(CDU)* und *Gerhard Schröder(SPD)* zugeordnet werden kann, wird bei der automatischen Auflösung der internen Ambiguitäten durch Einführung eines vagen Konzepts *Gerhard Schröder'* aufgelöst, da sich die beiden Konzepte nur in sehr wenigen Themen unterscheiden (*SPD*, *CDU*,...) und somit einen hohen thematischen Überschneidungsgrad aufweisen.

Dagegen weisen die Konzepte *Computervirus* und *Viren (Mikroorganismen)*, die beide von dem ambigen Lexikoneintrag *„Virus"* referenziert werden einen geringen thematischen Überschneidungsgrad auf. Somit würde für diesen Begriff ein neues ambiges Konzept *Virus** eingeführt werden, das über die Relation Amb mit beiden Konzepten verbunden ist und von dem Lexikoneintrag referenziert wird.

So führen bei der semantischen Indexierung Vorkommen von *„Gerhard Schröder"* in den Dokumenten immer auch zu Indexeinträgen zu den thematischen Überschneidungen *Persönlichkeiten der deutschen Geschichte*, *Politik*, ... Dabei gehen Information verloren, da nicht eindeutig geklärt wird, um welchen der beiden Politiker es sich handelt. Dennoch bleibt eine gute thematische Einordnung der entsprechenden Dokumente erhalten, ohne die Vorkommen disambiguieren zu müssen.

Vorkommen von *„Virus"* dagegen müssen entweder als ambig im Index markiert werden oder disambiguiert werden. Da allerdings bei diesem Begriff die thematische Überschneidung sehr gering ist, sollte die thematische Disambiguierung gut zwischen den beiden möglichen Konzepten unterscheiden können.

Nach der Normalisierung der Lexikoneinträge und der Behandlung der Ambiguitäten ist sichergestellt, dass jeder Lexikoneintrag genau ein (eindeutiges oder ambiges) Konzept der Wissensressource referenziert. Die Lexikoneinträge werden sortiert und es wird der minimierte $\mathcal{W}$-Automat nach dem in Kapitel 3.4.2 dargestellten Algorithmus erzeugt. Die eindeutigen Identifikationsnummern der Konzepte dienen dabei der eindeutigen Zuordnung von Konzepten und werden an den Finalzuständen des $\mathcal{W}$-Automaten gespeichert.

8.3.3. Serialisierung

Nachdem der Graph und der $\mathcal{W}$-Automat der Wissensressource aus den Eingabedateien erzeugt wurden[2], können die Tripel der Eingabedateien verworfen werden. Um ein schnelleres Laden der Anwendung zu ermöglichen, wird die interne Repräsentation der Wissensressource auf eine geeignete Weise serialisiert.

Die Serialisierung der Wissensressource der semantischen Indexierung ermöglicht es, die Serveranwendung schneller zu starten, da durch die Serialisierung verhindert wird, dass die aufwendigen Berechnungen, die zum Aufbau der Wissensressource und des Lexikon nötig sind, nicht mehrfach wiederholt werden müssen.

```
[...]
<skos:Concept rdf:about="efgt:saurishia">
  <skos:prefLabel>Saurishia</skos:prefLabel>
  <skos:altLabel>Echsenbeckensaurier</skos:altLabel>
  <skos:broader rdf:resource="efgt:dinosauria"/>
  <skos:narrower rdf:resource="efgt:velociraptor"/>
</skos:Concept>
<skos:Concept rdf:about=efgt:velociraptor>
  <skos:prefLabel>Veloziraptor</skos:prefLabel>
  <skos:altLabel>schneller Räuber</skos:altLabel>
  <skos:broader rdf:resource="efgt:saurishia"/>
</skos:Concept>
[...]
```

Beispiel 8.3.2. Die hier dargestellte, in RDF-XML kodierte, Wissensressource solle von Semix zur semantischen Indexierung verwendet werden. Die Konfiguration zu

[2]Das Lexikon der Wissensressource $\mathcal{W}_{Lex}$ wird von Semix bei der semantischen Indexierung nicht verwendet, da es nur zur Erzeugung des Lexikonautomaten benötigt wird. Semix speichert dennoch auch $\mathcal{W}_{Lex}$, um auch lexikalische Informationen über die Konzepte der expliziten Wissensressource ausgeben zu können.

dieser Ressource, spezifiziere die Prädikate *skos:prefLabel* und *skos:altLabel* als eindeutige Vorzugs- bzw. Alternativnamen und das Prädikat *skos:broader* als transitiv. Das Prädikat *skos:narrower* soll beim Aufbau ignoriert werden (vgl. Appendix B.2).

Beim Parsen der Ressource werden die Konzepte in eine Reihe von Tripel aufgespalten, aus denen dann die interne Repräsentation der Wissensressource aufgebaut wird.

Die Tripel, die Verbindungen zwischen Konzepten in der Wissensressource spezifizieren, werden eingelesen und nach ihren Prädikaten sortiert. In dem hier dargestellten Fall, verfügt die Wissensressource nur über die einzige Relation *skos:broader*, da die Verbindungen über die Relation *skos:narrower* ignoriert werden: *skos:broader* $=$ $\{\langle$*efgt:saurishia*, *efgt:dinosauria*$\rangle, \langle$*efgt:velociraptor*, *efgt:saurishia*$\rangle\}$.

Da das Prädikat *skos:broader* als transitive Relation markiert ist, wird nun dessen transitive Hülle berechnet und für die Relation selbst und jedes Konzept dieser Relation ein Knoten im Graphen erzeugt. Jeder Knoten kann dabei eindeutig über dessen URI und über eine eindeutige, interne Identifikationsnummer angesprochen werden. Aus allen relationalen Verbindungen in der transitiven Hülle der Relation *skos:broader* werden für die entsprechenden Konzepte die entsprechenden Kanten des Graphen generiert:

- *efgt:dinosauria* $\rightarrow 1 : \emptyset$
- *efgt:saurishia* $\rightarrow 2 : \{\langle 4, 1\rangle\}$
- *efgt:velociraptor* $\rightarrow 3 : \{\langle 4, 1\rangle, \langle 4, 2\rangle\}$
- *skos:broader* $\rightarrow 4 : \emptyset$

Nachdem der Graph aus den Eingabetripeln aufgebaut worden ist, werden aus allen verbleibenden Tripeln mit Namensprädikaten das Lexikon der Wissensressource aufgebaut. Die Lexikoneinträge werden normalisiert und über die URIs den Identifikationsnummern der Knoten im Graphen zugeordnet: $\{\langle$␣*saurishia*␣$, 2\rangle, \ldots, \langle$␣*velociraptor*␣$, 3\rangle, \ldots\}$.

Aus diesen Einträgen wird schließlich der $\mathcal{W}$-Automat aufgebaut. Die konzeptuellen Zuordnungen von Lexikoneinträgen zu Konzeptknoten geschieht dann einfach über die eindeutigen Identifikationsnummern.

Die beiden Strukturen der internen Repräsentation der Wissensressource können nun serialisiert werden, um einen erneuten Aufbau zu verhindern und um ein schnelleres Starten der Anwendung zu gewährleisten.

8.4. Aufbau des Index

Der Aufbau des Index in Semix folgt den Darstellungen zum Indexaufbau der semantischen Indexierung aus Kapitel 5.3. Die Dateinamen der Postingfiles leiten sich aus den eindeutigen URIs der zugeordneten Konzepte ab und liegen in einer zusätzlichen Struktur aus Unterverzeichnissen, um eine Verteilung der Postingfiles auf verschiedene Festplatten oder Rechner zu ermöglichen. So ist es auch möglich, verschiedene Indexe zu unterschiedlichen Wissensressourcen — sofern diese über unterschiedliche URIs verfügen — parallel zu verwalten.

Semix verwendet neben einem Relationsregister, in welchem die Identifikationsnummern der Prädikate in der Wissensressource gespeichert werden, ein Dokumentenregister (vgl. Kapitel 5.3.2) um die redundante Speicherung der Dokumentenpfade in jedem einzelnen Indexeintrag zu verhindern. Die Indexeinträge in den Postingfiles bestehen immer zumindest aus je einem Feld zur Speicherung des Eintragstyps und einem Feld zur Speicherung der Dokumentenidentifikationsnummern. Je nach Konfiguration der Software ist das Feld des Eintragstyps entweder eine einfache boolesche Variable[3] oder die Identifikationsnummer einer Relation des Graphen[4]. Weitere Konfigurationsoptionen steuern die Speicherung der Offsetpostionen der Textbelege sowie die Speicherung derselbigen Textbelege (vgl. Appendix A.3).

8.5. Indexierung

Zur Indexierung von Dokumenten mit Semix wird zuallererst eine Normalisierung der Eingabedokumente durchgeführt. Mit Hilfe des $\mathcal{W}$-Automaten wird die multiple Stringsuche, wie in Kapitel 5.4.1 dargestellt, auf den normalisierten Dokumenten ausgeführt. Jedem, während der Suche in einem Dokument gefundenen Textbeleg, wird sein entsprechendes Konzept und weitere, für die Weiterverarbeitung wichtige Informationen, wie etwa die Start- und Endposition im Dokument, zugeordnet. Alle Textbelege werden in genau der Reihenfolge, in der sie in den Dokumenten gefunden wurden, an die nachfolgenden Verarbeitungsschritte der semantischen Indexierung weitergereicht. Jeder Textbeleg enthält dabei alle nötigen Daten um ihn eindeutig in einen entsprechenden Indexeintrag für den semantischen Index umzuwandeln.

Damit weitere approximative Suchen hintereinander nachgeschaltet werden können (vgl. Kapitel 6.3.5), werden nicht nur Textbelege zu gefundenen Konzepten erzeugt, sondern auch zu den Textstellen zwischen den Ergebnissen der strik-

[3] Bei der Verwendung von EFGT-Netzen.
[4] Bei der Verwendung von Ontologien.

ten Suche, in denen keine Textbelege gefunden werden konnten. Diese, als *nicht-assoziierte Textstellen* bezeichneten, Textbelege verfügen über keine konzeptuelle Zuordnung und werden bei der Indexierung ignoriert. Nachgeschaltete Suchen arbeiten dann nur auf diesen nicht-assoziierten Textbelegen und überspringen bereits behandelte Textbelege (vgl. Kapitel 8.7).

Um sicherzustellen, dass Textbelege ohne konzeptuelle Zuordnung[5] weiterhin normalisiert sind (vgl. Kapitel 5.4.1), werden die Textbelege zwischen Suchergebnissen so aufgeteilt, dass die führenden und nachfolgenden Tokenbegrenzungszeichen von Suchtreffern, den entsprechenden, unassoziierten Textstellen zugeordnet werden. Textbelege zu Lexikoneinträgen sind daher nicht von Tokenbegrenzungszeichen umgeben; unassoziierte Textstellen dagegen schon. Weitere Suchen auf unassoziierten Textstellen können so ohne Mehraufwand weiterhin auf normalisiertem Text ausgeführt werden.

Bei der eigentlichen Indexierung, direkt nach den Suchschritten werden für jeden, in den Dokumenten vorgefundenen Textbeleg, die entsprechenden direkten und indirekten Indexeinträge erzeugt und in die jeweiligen Buffer der Indexknoten geschrieben. Jeder Indexknoten entspricht genau einem Konzept der Wissensressource und kann eindeutig über die internen Identifikationsnummern angesprochen werden, über das immer eindeutig das passende Postingfile im Index identifiziert werden kann. Indexknoten, deren Buffer während der Indexierung eine bestimmte Maximalgröße überschreiten, werden automatisch blockweise an die entsprechenden Postingfiles des semantischen Index angehängt.

Da jeder Textbeleg bei der Suche mit einer entsprechenden konzeptuellen Zuordnung versehen wird und den zugeordneten Konzeptknoten ihre Verbindungen im Graphen der Wissensressource zugeordnet sind, können alle direkten und indirekten Konzepte einfach über den zugeordneten Konzeptknoten berechnet werden.

Der Index für die Dokumentenkollektion wird von Semix inkrementell aufgebaut, indem einzelne Dateien nach und nach verarbeitet und die jeweiligen Textbelege in den Index geschrieben werden. Alle Schritte der Indexierung — die strikte und approximative Suche sowie die Disambiguierung und die eigentliche Indexierung — werden dabei nebenläufig ausgeführt [30]. Damit kann die Indexierung parallel auf mehreren Prozessoren ausgeführt werden.

Semix unterstützt zur Indexierung neben einfachen Textdateien eine Reihe weiterer Dateiformate, darunter HTML, XML, hOCR [8], TEI [58] und ALTO [59], welche eigenständig identifiziert und verarbeitet werden.

[5] Dies sind ja genau die Textstellen, auf denen noch nachgeschaltete Suchen durchgeführt werden könnten.

8.6. Suchanfragen

Semix unterstützt alle Möglichkeiten zu Suchanfragen, die in Kapitel 5.5 beschrieben wurden. Es werden einfache Suchen nach Textbelegen zu angefragten Konzepten, thematische Suchen und auch relationale Suchen unterstützt. Die Erweiterungen zur Suche von unscharfen und ambigen Suchtreffern, die in Kapitel 6.6 und Kapitel 7.3.3 erläutert wurden, werden ebenfalls unterstützt.

Semix setzt einen eigenen Parser für die Suchanfragen ein, der eine spezielle Syntax unterstützt, die der Syntax ähnelt, die auch zur Darstellung von Suchanfragen in dieser Arbeit verwendet wurde. Näheres zur Anfragesprache von Semix findet sich in Appendix B.3.

Die Konzepte und auch die Relationen in Suchanfragen können einerseits durch ihre eindeutigen URIs spezifiziert werden. Auch können die Konzepte durch natürlichsprachliche Ausdrücke identifiziert werden, indem der $\mathcal{W}$-Automat der Wissensressource zur Identifizierung von Lexikoneinträgen in den Suchanfragen verwendet wird. So können bei Suchanfragen sowohl die URIs als auch die Vorzugsnamen der angefragten Konzepte verwendet werden.

Die Abarbeitung der Suchanfragen geschieht dann genau, wie in Kapitel 5.5 dargestellt. Für jedes angefragte Konzept der Anfragemenge $q \in Q \subseteq \mathcal{W}_K$, wird das entsprechende Postingfile über dessen eindeutige URI im Index identifiziert und die Indexeinträge nacheinander aus der Datei ausgelesen. Dabei werden schon beim Lesen der Indexeinträge, je nach genauer Suchanfrage, indirekte, fehlerbehaftete oder ambige Indexeinträge aus der Ergebnismenge herausgefiltert und nur die entsprechenden Indexeinträge zurückgeliefert.

Semix liest nicht nur die Indexeinträge der Postingfiles aus, sondern auch die Indexeinträge, die sich gegebenenfalls noch in einem internen Buffer befinden. Sobald ein Dokument von Semix prozessiert wurde, stehen die entsprechenden Indexeinträge für Suchanfragen zur Verfügung, egal ob sie sich noch in einem internen Buffer befinden, oder schon in ein Postingfile geschrieben wurden.

Alle Indexeinträge der Ergebnismenge von Suchanfragen enthalten die gesamte im Index gespeicherte Information. Je nach verwendeter Konfiguration des Index (vgl. Appendix A.3) können die zurückgelieferten Indexeinträge unterschiedliche Informationen zu den Textbelegen, Positionen und relationalen Verbindungen enthalten. Unabhängig davon wie der Index genau konfiguriert ist, enthalten alle Indexeinträge zumindest Informationen darüber, in welchem Dokument sie gefunden wurden, ob es sich um direkte, indirekte oder ambige Einträge handelt, mit welchem genauen Fehler der entsprechende Eintrag im Dokument gefunden wurde und ob es sich um einen ambigen oder eindeutigen Indexeintrag handelt.

8.7. Fehlertolerante Suche

Die in Kapitel 6 dargestellte fehlertolerante Suche von Lexikoneinträgen ist ebenfalls in Semix realisiert. Wie bereits weiter oben erwähnt wurde, wird der minimierte Automat durch eine verschränkte Übergangstabelle dargestellt und enthält alle Informationen, um auch die fehlertolerante Suche zu realisieren. Der $\mathcal{W}$-Automat wird von Semix sowohl zur strikten als auch zur unscharfen Suche mit einer festen Obergrenze von k verwendet werden.

Semix verwendet UTF-8 als internes Kodierungsformat (siehe oben). Somit entsprechen einzelne Übergänge nicht immer einem tatsächlichen Druckbuchstaben, sondern können auch Teil einer Multibyte-Sequenz zur Darstellung einzelner Buchstaben aus höheren Unicode-Ebenen sein. So könnte es bei einer fehlertolerante Suche dazu kommen, dass der Unterschied zweier (Druck-) Buchstaben zu mehr als einem Levenshteinabstand von 1 führt. Aus diesem Grund behandelt die fehlertolerante Suche — wie sie in Semix implementiert ist — solche Multibyte-Sequenzen und stellt sicher, dass es nicht zu solchen falschen Berechnungen der Levenshteindistanz kommen kann.

Beispiel 8.7.1. Die Kodierung des Buchstaben *e* ist in UTF-8 in hexadezimaler Notation `0x65`. Die des Buchstaben *ë* dagegen die Multibyte-Sequenz `0xC30xAB`. Mit einer naiven *byte-weisen* Berechnung der Levenshteindistanz würde für den Abstand zwischen den beiden Token *Semix* und *Sëmix* fälschlicher Weise $d_L(\textit{Semix}, \textit{Sëmix}) = 2$ gelten.

Neben diesen technischen Problemen der verwendeten Eingabekodierungen, können auch sogenannte *combining characters*[6] [23, S.111ff.] zu einem ähnlichen Effekt führen. Diese kombinierenden Zeichen sind komplexe Druckzeichen[7], die als mehrere Unicode-Punkte dargestellt werden. Auftreten solcher kombinierter Zeichenfolgen — in Kombination unterschiedlicher Kompositions- und Dekompositionsverfahren im Unicode-Standard — führen bei der unscharfen Suche zu ähnlichen Effekten wie beim Auftreten von Multibyte-Sequenzen in UTF-8. Diese Effekte werden von Semix nicht behandelt. Aus diesem Grund sollte bei der Verwendung der unscharfen Suche — und auch bei der Verwendung von Semix im Allgemeinen — darauf geachtet werden, dass die verwendeten Wissensressourcen und Eingabedokumente auf dieselbe Weise normalisiert worden sind.

Zur fehlertoleranten Suche auf Dokumenten können der strikten Suche bei der Indexierung von Dokumenten weitere unscharfe Suchen mit aufsteigenden Obergrenzen von $k_1, \ldots, k_n$ mit $k_1 < \cdots < k_n$ angehängt werden. Eine strikte Suche

[6] Deutsch: kombinierende Zeichen.

[7] Zumeist einfache Zeichen kombiniert mit verschiedenen Akzentuierungen.

ist dabei obligatorisch. Es ist nicht möglich nur unscharfe Suchen bei der semantischen Indexierung zu verwenden. Die deutlich rechenintensivere unscharfe Suche soll wirklich nur fehlerbehaftete Suchergebnisse behandeln und nicht indirekt für strikten Suchen missbraucht werden. (vgl. Kapitel 6.3.5). Diese nachgeschalteten Suchen laufen dabei immer nur auf den Teilen der Dokumente, auf denen vorangegangene Suchdurchläufe keine Ergebnisse liefern konnten.

Wie bereits in Kapitel 8.5 dargestellt, werden bei der strikten Suche nicht nur die Textbelege zu gefundenen Lexikoneinträgen erzeugt, sondern auch zu den Textteilen in denen keine Einträge gefunden werden konnten. Diese Textbelege verfügen noch über keine konzeptuelle Zuordnung und alle nachgeschalteten Suchen laufen nur auf diesen Textbelegen ab.

Werden bei einer unscharfen Suche auf einem Textbeleg Lexikoneinträge gefunden, wird dieser wiederum in Teile assoziierte und nicht-assoziierte Textteile mit und ohne konzeptuelle Zuordnungen aufgespalten. Nachgeschaltete unscharfe Suchen werden dann ihrerseits nur auf Textbelegen ohne konzeptuelle Zuordnungen angewandt. So wird sichergestellt, dass immer nur Teile der Eingabedokumente durchsucht werden, auf denen vorangegangene Suchen keine Ergebnisse lieferten. Semix stellt dabei sicher, dass die Reihenfolge der Textbelege erhalten bleibt, um bei einer eventuell nachgeschalteten Disambiguierung einen validen Dokumentenkontext zu erhalten (vgl. Kapitel 8.8).

Alle Textbelege die bei der fehlertoleranten Suche gefunden werden können, werden mit ihrem entsprechenden Fehler markiert, indem ein temporäres, anonymes Konzept (vgl. Kapitel 6.3.6) erzeugt wird, welches die eigentlich vorgefundenen Konzepte mit einem entsprechenden Fehler referenziert.

In Semix werden alle Konzepte, die bei der fehlertolerante Suche gefunden werden indirekt über anonyme Konzepte referenziert, egal ob es sich bei dem gefundenen Textbeleg um eine Mehrdeutigkeit der unscharfen Suche handelt oder nicht. Dies verallgemeinert die Behandlung von fehlerbehafteten und ambigen Textbelegen und erleichtert die weiteren Verarbeitungsschritte. Es können aber weiterhin anonyme Konzepte von Mehrdeutigkeiten und eindeutige Zuordnungen unterschieden werden. Erstere referenzieren mehr als ein Konzept, letztere genau eines.

Die anonymen Konzepte werden bei der Erzeugung der Indexeinträge gesondert behandelt, wobei die beiden Arten von anonymen Konzepten nicht weiter unterschieden werden müssen. Nicht für die anonymen Konzepte selbst, sondern für deren referenzierte Konzepte werden auf die übliche Weise direkte und indirekte Indexeinträge mit einem entsprechenden Fehler generiert. Diese werden dann in die jeweiligen Postingfiles des Index geschrieben.

Alle Indexeinträge von fehlerbehafteten Textbelegen tragen einen entsprechenden Fehler. Alle Indexeinträge mit einem Fehler werden bei normalen Suchan-

fragen ausgefiltert. Erst wenn bei Suchanfragen explizit eine Obergrenze für den maximal zulässigen Fehler angegeben ist, werden fehlerbehaftete Indexeinträge in die Ergebnismenge von Suchanfragen aufgenommen.

8.8. Behandlung von Ambiguitäten

Ambiguitäten werden von Semix schon beim Aufbau der Wissensressource behandelt. Lexikoneinträge, die externe Ambiguitäten darstellen, müssen über speziell ausgezeichnete Relationen (vgl. Appendix B.2) markiert sein. Interne Ambiguitäten dagegen müssen nicht extra in den Wissensressourcen ausgezeichnet sein und werden von Semix automatisch behandelt.

Über die Konfiguration der Wissensressource (vgl. Appendix B.2), kann eingestellt werden wie interne Ambiguitäten von Semix behandelt werden sollen. Ambige Lexikoneinträge können entweder verworfen, in die Wissensressource übernommen oder aufgelöst werden (vgl. Kapitel 7.2.2). Es steht auch eine Option zur Verfügung, die Übernahme und Auflösung interner Ambiguitäten für einzelne ambige Lexikoneinträge auf Basis einer einfachen Heuristik automatisch auszuwählen (vgl. Kapitel 8.3.2).

Bei der Suche nach Konzepten in den Dokumenten müssen ambige Konzepte interner und externer Ambiguitäten nicht gesondert behandelt werden. Jedem Lexikoneintrag ist weiterhin genau ein Konzept zugeordnet, welcher bei der Suche dem gefundenen Textbeleg zugeordnet wird. Wie bereits beschrieben, werden ambige Konzepte erst bei der Indexerzeugung gesondert behandelt. Wie auch schon bei der fehlertoleranten Suche, wird nicht ein ambiges Konzept, sondern die von diesen referenzierten Konzepte nach dem bekannten Schema indexiert und mit einer speziellen Markierung als ambige Textbelege im Index ausgezeichnet. Aufgrund dieser Markierung werden sie bei normalen Suchen nicht in die Ergebnismenge aufgenommen. Sie können nur durch tolerante Suchanfragen gefunden werden.

Während der Indexierung kann auch versucht werden, ambige Konzepte zu disambiguieren. Hierzu können beliebige verschiedene Methoden zur Ambiguitätsauflösung zwischen den Suchvorgang und die Indexerzeugung geschaltet werden. Semix unterstützt alle drei, der in Kapitel 7.4 dargestellten Methoden zur Ambiguitätsauflösung. Alle drei dieser Methoden verwenden intern ein lokales, dokumentenabhängiges Gedächtnis mit frei konfigurierbarer Maximalgröße. Das Gedächtnis wird während der Disambiguierung mit allen nicht ambigen Konzepten befüllt, so dass während der Disambiguierung immer der aktuelle thematische Dokumentenkontext zur Verfügung steht.

Die beiden automatischen Methoden zur Disambiguierung — die einfache (vgl. Kapitel 7.4.4) und die thematische (vgl. Kapitel 7.4.6) Disambiguierung — sind ein-

fach implementiert, indem sie die benötigten Berechnungen für die zu disambiguierenden Konzepte auf dem Gedächtnis ausführen und dann entweder ein Konzept eindeutig disambiguieren oder eben scheitern. Sie können dem Indexierungsprozess ohne zusätzlichen manuellen Aufwand hinzugefügt werden.

Die regelbasierte Disambiguierung dagegen benötigt zur Disambiguierung konzeptabhängige Regeln, die schon beim Aufbau der Wissensressource den jeweiligen Konzepten zugeordnet sein müssen. Hierzu können einzelnen Konzepten der Wissensressource über ein spezielles Prädikat, welches in der Konfiguration der Wissensressource (vgl. Appendix B.2) ausgezeichnet sein muss, zugeordnet werden. Beim Aufbau der internen Wissensressource werden diese Regeln geparst, in ein internes Format umgewandelt und ihren entsprechenden Konzepten zugeordnet. Die Regeln verfügen dabei über verschiedene arithmetische Operatoren und die in Kapitel 7.4.5 dargestellten Funktionen, um auf den Dokumentenkontext zugreifen zu können. Näheres zur Syntax der Disambiguierungsregeln findet sich in Appendix B.4.

Bei der regelbasierten Disambiguierung werden diese kompilierten Regeln der an der Ambiguität beteiligten Konzepte mit dem aktuellen Gedächtnis ausgeführt. Die kompilierten Regeln sind intern als eine einfache Stack-Maschine repräsentiert, mit der die nötigen Berechnungen relativ effizient ausgeführt werden können. Wie bereits beschrieben wird dann genau das Konzept disambiguiert, dessen Regel als einziges *wahr* zurück liefert. Falls keine oder mehrere Regeln der beteiligten Konzepte *wahr* zurückliefern, scheitert die regelbasierte Disambiguierung.

Falls eine der verwendeten Methoden zur Disambiguierung erfolgreich ist, wird das disambiguierte Konzept in das Gedächtnis geschrieben und die Referenz des Textbeleges so angepasst, dass nicht mehr das ambige Konzept sondern das eindeutige Konzept referenziert wird. Hierzu muss nur die konzeptuelle Referenz des Textbeleges von dem ambigen Konzept auf das entsprechende eindeutige Konzept umgeleitet werden. Für alle späteren Verarbeitungsschritte, inklusive der Indexierung und anderer nachgeschalteter Disambiguierungsverfahren, handelt es sich dann um einen normalen eindeutigen Textbeleg, welcher auf ein eindeutiges, nicht ambiges Konzept verweist. Dieser kann dann ohne weitere Sonderbehandlung weiter verarbeitet werden.

8.9. Auswertungen

Es folgen nun eine Reihe von Auswertungen der semantischen Indexierung. Für alle Auswertungen wurde immer die vorliegende Implementierung von Semix verwendet. Je nach Auswertung wurde immer entweder ein Ausschnitt des TopicZoom-Netzes oder der Ontologie „Erster Weltkrieg“ verwendet.

Der Ausschnitt des TopicZoom-Netzes umfasst dabei 10.334 Konzepte, die von 24.770 Lexikoneinträgen referenziert werden. Die Ontologie „Erster Weltkrieg“ umfasst 10.694 verschiedene Konzepte und 223.970 Lexikoneinträge in den drei Sprachen Deutsch Englisch und Russisch (vgl. Kapitel 3.5).

Als Korpus für den Dokumentenindex wurden jeweils komprimierte XML-Dumps[8] der Deutschen, Englischen und Russischen Wikipedia aus dem Jahr 2016 verwendet. Dabei wurden immer kurze Artikel mit weniger als 100 Zeichen und ebenso verschiedene Spezialseiten der Wikipedia, wie unter anderem Bedeutungsklärungen, Listen, usw..., ignoriert.

Für Auswertungen mit EFGT-Netzen wurde nur die Deutsche Wikipedia, für die Auswertungen mit der Ontologie „Erster Weltkrieg“ wurden dagegen alle drei verschieden-sprachlichen Korpora verwendet. Hierbei wurden immer nur Artikel indexiert, deren Wikipedia-Kategorie auf das Themengebiet des Ersten Weltkrieg verweisen[9].

Alle Auswertungen wurden auf einem zeitgemäßen Desktop-Rechner mit 8 GB RAM mit 4 Intel(R) Core(TM) i5 CPU's mit jeweils 2,50 GHz ausgeführt. Das Indexverzeichnis wurde dabei auf einer externen USB-Festplatte gespeichert. Bei allen Ausführungen der Auswertungen liefen sowohl die Server- als auch die Client-Anwendung auf demselben Rechner und kommunizierten jeweils über einfache TCP-IP Sockets.

Ziel der Auswertungen war es nicht den optimierten Fall auszuwerten, sondern ein möglichst realistisches Bild der semantischen Indexierung mit expliziten Wissensressourcen, wie es in der aktuellen Implementierung vorliegt, darzustellen. So werden die einzelnen Wikipedia-Artikel aus dem XML-Dump geparst und auf den Server als einfache Textdatei hochgeladen. Hierbei muss der Server aus diesen hochgeladenen Dateien eine eigenständige Kopie in einem speziellen Dump-Verzeichnis erzeugen, um die jeweiligen Dateien für spätere Suchen und zur Darstellung der Textbelege bereitstellen zu können. Auch wurde eine externe USB-Festplatte zur Speicherung des semantischen Index verwendet

8.9.1. Evaluierung der grundlegenden semantischen Indexierung

Zur Auswertung der Indexierung wurden jeweils $N = 10, 100, \ldots, 100.000$ Wikipedia-Artikel indexiert und die Zeiten der Indexierung gemessen. Jeder Indexierungsvorgang wurde dabei 10 mal wiederholt und dann der Mittelwert berechnet.

[8] `https://de.wikipedia.org/wiki/Wikipedia:Technik/Datenbank/Download`

[9] Die Wikipedia-Kategorien *Erster Weltkrieg*, *world war I*, *first world war* und *Первая мировая война*.

N	Wissensressource	Zeit	Indexgröße	Indexeinträge
10	TopicZoom	0.33 s	14 MB	742
100	TopicZoom	1.49 s	25 MB	13.309
1000	TopicZoom	10.88 s	153 MB	106.080
10.000	TopicZoom	106.72 s	217 MB	1.014.308
100.000	TopicZoom	569.86 s	853 MB	4.455.698
10	„Erster Weltkrieg“	1.10 s	0.5 MB	215
100	„Erster Weltkrieg“	3.73 s	4 MB	3144
1000	„Erster Weltkrieg“	28.80 s	11.5 MB	28.500
10.000	„Erster Weltkrieg“	150.10 s	33 MB	203.954
100.000	„Erster Weltkrieg“	758.41 s	141 MB	1.548.931

Tabelle 8.1.: Grundlegende semantische Indexierung mit einem TopicZoom-Netz und der Ontologie „Erster Weltkrieg“

Alle hier dargestellten Indexierungszeiten beinhalten immer auch das Leeren aller internen Buffer auf die Festplatte und geben somit die Zeiten an, die benötigt werden, um den gesamten Index auf die Festplatte zu schreiben.

Es wurde auch die Größe des erzeugten Index gemessen. Hierzu wurde das einfache Hilfsprogramm du[10] verwendet. Die angegebenen Indexgrößen beziehen sich nur auf den erzeugten Dokumentenindex und nicht auf die Größe der erzeugten Dump-Dateien (siehe oben).

Tabelle 8.1 fasst die Ergebnisse der Indexierung von Wikipedia-Artikeln mit dem TopicZoom-Netz und der Ontologie „Erster Weltkrieg“ zusammen. Dabei ist ersichtlich, dass die Dauer der Indexierung für beide Wissensressourcen annähernd linear abhängig von der Anzahl der Artikel ist. Dies war zu erwarten, da ja die semantische Indexierung den Index direkt dokumentenweise aufbaut und keine weiteren Zwischenschritte bei der Indexierung vornimmt.

Die Zeiten für beide ausgewerteten Wissensressourcen sind vergleichbar, wobei die Indexierung mit der Ontologie „Erster Weltkrieg“ etwas langsamer war. Dies ist wohl vor allem der rigorosen Auswahl der Artikel geschuldet, da nicht nur ein Korpus, sondern drei Korpora auf der Suche nach geeigneten Artikeln weiter durchlaufen werden müssen als das Korpus für die EFGT-Netze, die allgemeinere Themengebiete umfassen. So zeigt sich auch ein geringer Einfluss der Größe der verwendeten Wissensressourcen auf die Indexierungszeiten der semantischen Indexierung.

Des weiteren scheint die Größe des semantischen Index, wie es zu erwarten

[10] `http://www.man7.org/linux/man-pages/man1/du.1.html`

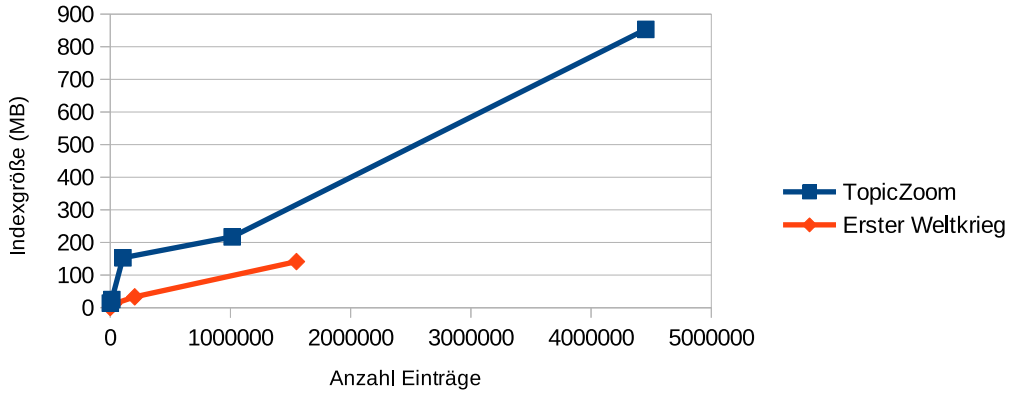


Abbildung 8.1.: Zusammenhang zwischen der Indexgröße des resultierenden semantischen Index und der Anzahl der in den Artikel gefundenen Textbelege. Die blaue Linie bezeichnet das TopicZoom-Netz, die orange Linie die Ontologie „Erster Weltkrieg“.

war, stärker von der Anzahl der identifizierten Textbelege in den Artikeln als von der Anzahl der indexierten Artikel abzuhängen (vgl. Abbildung 8.1). Dabei scheint die Indexgröße ebenfalls linear abhängig von der Anzahl der Textbelege in den Artikeln zu sein.

Beim Vergleich der Indexgrößen der beiden Wissensressourcen zeigt sich, dass der semantische Index für EFGT-Netze deutlich größer ist, als für die Ontologie „Erster Weltkrieg“. Dies liegt einfach in der Tatsache begründet, dass die Ontologie — im Gegensatz zum TopicZoom-Netz — weniger transitive Verbindungen aufweist. Wie bereits dargestellt wurde, sammeln die höherliegenden und damit allgemeineren Konzepte[11] in EFGT-Netzen aufgrund der Transitivität der zugrundeliegenden Relation in den EFGT-Netzen sehr viele Indexeinträge an. Dies spiegelt sich auch direkt in den resultierenden Indexgrößen für die TopicZoom-Netze und die Ontologie „Erster Weltkrieg“ wieder.

8.9.2. Evaluierung von Suchanfragen

Um auch den Einfluss der Indexgröße auf die Dauer von Suchanfragen zu untersuchen, wurden jeweils 1000 unterschiedliche einfache und thematische bzw. relationale Suchanfragen zu verschiedenen Konzepten auf mehreren semantischen Indexen mit verschiedenen Größen durchgeführt. Als Konzepte für die verwendeten

[11] Das Postingfile des Konzept *Topnode*, welches ja das allgemeinste Konzept der Hierarchie darstellt, enthält *alle* Indexeinträge des gesamten semantischen Index.

	TopicZoom-Netz		Ontologie „Erster Weltkrieg“	
N	einfach	thematisch	einfach	thematisch
10	8.85 s	9.07 s	10.18 s	9.86 s
100	9.33 s	10.13 s	9.97 s	9.98 s
1000	10.75 s	15.28 s	10.03 s	10.01 s
10.000	20.32 s	58.17 s	10.23 s	10.55 s
100.000	48.11 s	204.32 s	11.91 s	12.58 s

Tabelle 8.2.: Anfragezeiten für jeweils 1000 einfache und 1000 thematische bzw. relationale Suchen auf einem semantischen Index.

Suchanfragen wurden verschiedene Lexikoneinträge aus den jeweiligen Wissensressourcen extrahiert und für die jeweiligen einfachen und thematischen Suchanfragen verwendet. Wie auch schon vorher wurden die Experimente jeweils 10 Mal wiederholt und die angegebenen Zeiten beziehen sich immer auf die Mittelwerte aus 10 Experimenten.

Die Ergebnisse dieser Auswertung sind in Tabelle 8.2 zusammengefasst. Für Suchanfragen mit der Ontologie „Erster Weltkrieg“ ergibt sich ein einheitliches Bild: die Größe des semantischen Index hat nur einen sehr geringen Einfluss auf die Anfragezeiten. Auch unterscheiden sich die Anfragezeiten relationaler Suchanfragen nicht von denen einfacher Suchanfragen. Für $N = 100.000$ ergibt sich eine leichte Vergrößerung der Suchanfragezeiten. Dies liegt wohl daran, dass für diese Indexgröße deutlich mehr Resultate zurückgeliefert und verarbeitet werden müssen.

Für die TopicZoom-Netze dagegen ergibt sich ein anderes Bild. Die Suchanfragezeiten steigen mit zunehmender Indexgröße an. Ebenso benötigen thematische Suchanfragen mehr Zeit als einfache. Da sich die Behandlung von Suchanfragen auf einem semantischen Index aus EFGT-Netzen nicht von der Behandlung von Suchanfragen auf einem semantischen Index aus Ontologien unterscheidet, müssen die abweichenden Ergebnisse über die Struktur der verwendeten Wissensressourcen erklärt werden.

Wie bereits mehrfach erwähnt wurde und wie es sich auch in den Indexgrößen widerspiegelt (vgl. Tabelle 8.1), weisen EFGT-Netze und damit das TopicZoom-Netz eine spezielle hierarchische Struktur auf, die sich stark von der Struktur der Ontologie „Erster Weltkrieg“ unterscheidet. Allgemeine Konzepte, befinden sich sehr weit oben in der Hierarchie und werden von vielen Unterthemen referenziert. Somit sind die Indexdateien von allgemeineren Konzepten viel größer als die von speziellen Konzepten.

Hierbei müssen dann für Suchanfragen zu solchen allgemeineren Konzepten deutlich größere Dateien verarbeitet werden[12]. Für thematische Suchanfragen müssen gegebenenfalls nicht nur größere Dateien verarbeitet werden, sondern auch — neben den direkten Indexeinträgen — auch alle indirekten Indexeinträge in den Dateien zurückgeliefert werden. Es ergeben sich somit viel mehr Ergebnisse, die verarbeitet und zurückgeliefert werden müssen.

Um Suchanfragezeiten auch nach Konzepten mit großen Indexdateien stabil halten zu können, ist es möglich, die Suchergebnisse auch blockweise zu lesen. Hierzu können Suchanfragen mit Semix so konfiguriert werden, dass nur feste Blöcke mit einer maximalen Anzahl von Suchergebnissen zurückgeliefert werden (vgl. Appendix B.5.3).

Die hier dargestellten Anfragezeiten von Suchanfragen lassen sich mit der Struktur der jeweils verwendeten Wissensressource erklären. Die Anfragezeiten liefern darüber hinaus auch ein Bild, wie der resultierende Index aufgebaut ist. Im Falle von EFGT-Netzen mit den stark hierarchischen Strukturen liegen deutlich größer Indexdateien vor, wohingegen im Falle von Ontologien mit weniger stark formalisierten Strukturen die Informationen auf unterschiedliche Indexdateien mit ähnlichen Größen verteilt sind.

8.10. Zusammenfassung

In Semix wurden alle Aspekte der semantischen Indexierung mit expliziten Wissensressourcen, von den verwendeten Automaten zur Repräsentation der Lexika, über die Erzeugung des semantischen Index, hin zur Behandlung von Ambiguitäten, wie sie in dieser Arbeit dargestellt wurden, implementiert. Die Implementierung von Semix ist dabei möglichst einfach gehalten und es wurde darauf verzichtet, verschiedene Optimierungsmöglichkeiten zu implementieren.

Dennoch ist diese einfache Implementierung der semantischen Indexierung dazu in der Lage, auch größere Datenmengen in vertretbarer Zeit semantisch zu indexieren. Die Anwendung kann über verschiedene Konfigurationsoptionen an verschiedene Wissensressourcen angepasst werden. So können eine Vielzahl unterschiedlicher Wissensressourcen zur semantischen Indexierung verwendet werden. Darüber hinaus können die von Semix verwalteten Indexe sequenziell aufgebaut werden und auch verschiedene Wissensressourcen parallel verarbeitet werden.

Durch die Client- Server-Architektur kann Semix, wie in Appendix B dargestellt, relativ einfach verwendet werden. Der Serverprozess verwaltet dabei die entspre-

[12] Da die vorliegenden Indexdateien einzelner Konzepte nicht strukturiert sind, muss auch bei der Bearbeitung einfacher Suchanfragen immer die gesamte Indexdatei nach direkten Einträgen durchsucht werden.

chenden expliziten Wissensressourcen und den semantischen Index. So können die verschiedenen Ressourcen global vorgehalten werden und müssen nicht jedes mal neu erzeugt bzw. eingelesen werden. Die einzelnen Operationen auf dem semantischen Index können dann mit einfachen Programmaufrufen durchgeführt werden, die auch dazu geeignet sind, über Skripte automatisiert zu werden, wie dies auch im Falle der dargestellten Auswertungen in diesem Kapitel geschehen ist.

Da Semix neben den reinen expliziten Wissensressourcen auch verschiedene zusätzliche Informationen über die Wissensressourcen vorhält, kann die Anwendung auch zur interaktiven Verwendung der Wissensressourcen verwendet werden. Hierzu enthält Semix ein einfaches Web-Frontend, mit dem die expliziten Wissensressourcen und der semantische Index mit Hilfe eines Web-Browsers interaktiv erkundet werden können (vgl. Appendix B.5.7).

9. Fazit

In dieser Arbeit wurde die semantische Indexierung mit expliziten Wissensressourcen mit alle ihren Aspekten behandelt. Es wurde gezeigt, wie ein einfacher Index unter Verwendung geeigneter expliziter Wissensressourcen mit semantischer Information angereichert werden kann und wie ein auf diese Weise angereicherter Index abgefragt werden kann. Hierzu wurden verschiedene linguistische und algorithmische Aspekte zum Aufbau eines semantischen Index beleuchtet. Ebenso wurden Erweiterungen zur fehlertoleranten Suche und im besonderen Maße auch die Behandlung von Ambiguitäten in den Wissensressourcen und bei der Indexierung diskutiert.

Die in Kapitel 1 gegebene Definition 1.5.1 der expliziten Wissensressourcen bildet die Basis für alle weiteren Betrachtungen der semantischen Indexierung. Explizite Wissensressourcen im Sinne dieser Arbeit bestehen aus einer Menge abstrakter Konzepte, einer Menge binärer Relationen auf diesen und einem Lexikon, welches über Konzeptzuweisungen Lexikoneinträge auf abstrakte Konzepte abbildet.

Die Grundlage der semantischen Indexierung bildet das Auffinden von Lexikoneinträgen in natürlichsprachlichen Texten. Hierzu wurden sogenannte $\mathcal{W}$-Automaten eingeführt (vgl. Definition 1.5.3), welche zur Suche der Lexikoneinträge von Konzepten eingesetzt werden.

Kapitel 3 stellte dazu ein Verfahren vor, mit dem es möglich ist solche $\mathcal{W}$-Automaten zu minimieren, ohne dabei die konzeptuellen Zuordnungen — welche eben der Abbildung von Lexikoneinträgen auf Konzepte dienen — zu verlieren. So können nicht nur die Präfixe der Lexikoneinträge, sondern auch geeignete Suffixe verschiedener Lexikoneinträge zusammengefasst werden. Dies führt insgesamt zu einer Reduktion der Zustandsmenge im Automaten (vgl. Tabelle 3.2) und somit auch zu einer Reduktion des Speicherbedarfs der verwendeten Lexika. Das Verfahren, welches von der semantischen Indexierung mit expliziten Wissensressourcen zur Suche von Konzepten in natürlichsprachlichen Dokumenten mit Automaten eingesetzt wird, wurde dann etwas später in Kapitel 5.4.1 dargestellt.

In Kapitel 4 wurden verschiedene Formen von Wissensressourcen diskutiert und wichtige Gemeinsamkeiten herausgearbeitet, auf deren Grundlage eine uniforme

Modellierung tatsächlicher Wissensressourcen im Sinne der expliziten Wissensressourcen bei der semantischen Indexierung geschehen kann. Hierbei sollte aber immer beachtet werden, dass zwar die Wissensressourcen uniform behandelt werden können, es sich aber durchaus — je nachdem wie die verwendeten Wissensressourcen genau aufgebaut sind — Unterschiede für verschiedene Aspekte der semantischen Indexierung ergeben können. Hierunter fallen unter anderem die verschiedenen Möglichkeiten von Suchanfragen und insbesondere auch die Auflösung von Ambiguitäten.

Aufbauend auf den Betrachtungen zu den verwendeten Wissensressourcen und $\mathcal{W}$-Automaten wurde dann in Kapitel 5 das Grundverfahren zur semantischen Indexierung mit expliziten Wissensressourcen dargestellt. Dabei wurde neben der Suche von Lexikoneinträgen auf Dokumenten (siehe oben), vor allem auch auf die Form des resultierenden semantischen Index und auf die unterschiedlichen Formen von Suchanfragen auf dem Index eingegangen.

Der eigentliche Index der semantischen Indexierung besteht dabei aus einer Menge von Postingfiles, wobei jedem Konzept der expliziten Wissensressource genau ein Postingfile zugeordnet ist. Jedes Postingfile enthält dabei alle direkten und indirekten Textbelege des zugeordneten Konzepts. Somit können alle (direkten und indirekten) Textbelege zu Suchanfragen eines Konzepts immer aus genau einer Datei ausgelesen werden.

Suchanfragen an den Index können dabei einerseits einfach nur nach Textbelegen zu gefundenen Konzepten suchen, oder andererseits thematisch aufgefasst werden (vgl. Kapitel 5.5). Im letzteren Fall werden auch Textbelege zu Konzepten zurückgeliefert, die mit den angefragten Konzepten über irgendeine Relation der Relationsmenge der expliziten Wissensressourcen verbunden sind. Dabei können — insbesondere bei der Verwendung von Ontologien — die relationalen Verbindungen der expliziten Wissensressourcen für gezielte Suchen auf der Dokumentenkollektion ausgenutzt werden und bestimmte relationale Verbindungen der Konzeptmenge verwendet oder ausgeschlossen werden.

Kapitel 6 behandelt wie auch fehlerbehaftete und historische Dokumente ohne einheitliche Orthographie semantisch indexiert werden können. Dabei wurde die fehlertolerante bzw. unscharfe Suche mit einer Fehlerschranke k auf dem Lexikonautomaten dargestellt, mit der es möglich ist, natürlichsprachliche Ausdrücke — insbesondere auch Mehrwortverbindungen — mit einem gewissen maximalen Levenshteinabstand in Dokumenten zu identifizieren. So können bei der semantischen Indexierung auch Konzepte auf fehlerbehafteten oder historischen Dokumenten gefunden und entsprechend indexiert werden.

Da die Ergebnisse der fehlertoleranten Suche nicht notwendigerweise richtig sind und da durch die fehlertolerante Suche auch mehrdeutige Konzeptzuweisungen entstehen können, werden die Ergebnisse fehlertoleranter Suchen im Index

markiert. Sie werden bei der Bearbeitung normaler Suchanfragen ignoriert und müssen daher durch fehlertolerante Suchanfragen explizit freigeschaltet werden.

Als letzter Aspekt der semantischen Indexierung mit expliziten Wissensressourcen wurde in Kapitel 7 die Behandlung von Ambiguitäten diskutiert. Dabei wurde zwischen drei unterschiedlichen Teilaspekten unterschieden.

Der erste Teilaspekt behandelte ambige natürlichsprachliche Ausdrücke in den Wissensressourcen, die der Identifikation von Konzepten in den Eingabedokumenten dienen. Die Konzepte der Wissensressourcen sind immer eindeutig. Die natürlichsprachlichen Begriffe, die diese referenzieren, sind dies im Allgemeinen nicht. Daher müssen ambige Begriffe bei der semantischen Indexierung auf die eine oder andere Weise behandelt werden. Hierzu ergeben sich — neben der Behandlung externer ambiger Ausdrücke, welche aus offensichtlichen Gründen manuell in den Wissensressourcen ausgezeichnet sein müssen — zwei Möglichkeiten. Ambiguitäten können entweder durch die Einführung neuer ambiger Konzepte mit in die Wissensressourcen übernommen werden oder beim Aufbau der Wissensressourcen durch die Einführung neuer vager Konzepte aufgelöst werden.

Nach der Behandlung ambiger natürlichsprachlicher Begriffe in den Wissensressourcen, wurde die Behandlung ambiger Konzepte bei der semantischen Indexierung erläutert. Wie fehlerbehaftete Textbelege auch, werden Textbelege zu ambigen Konzepten in den Postingfiles ausgezeichnet. Hierbei werden keine Indexeinträge in den Postingfiles der *ambigen* Konzepte, sondern in den Postingfiles der von diesen referenzierten *eindeutigen* Konzepten erzeugt. Ambige Textbelege tauchen ebenfalls nicht in normalen Suchanfragen auf, sondern werden nur bei toleranten Suchanfragen mit zurückgeliefert.

Als letzter Teilaspekt der Behandlung von Ambiguitäten, wurden drei unterschiedliche Vorgehen zur Disambiguierung vorgestellt, welche es auf der Basis der expliziten Wissensressourcen ermöglichen, Ambiguitäten aufzulösen. Neben einem naiven Verfahren, welches einfach nur die an der Ambiguität beteiligten eindeutigen Konzepte im Gedächtnis zählt und vergleicht, wurde zwei weiteren Verfahren zur Auflösung von Ambiguitäten dargestellt.

Zum einen können mittels der regelbasierten Ambiguitätsauflösung (vgl. Kapitel 7.4.5) manuelle Regeln zur Disambiguierung einzelner Konzepte formuliert werden. Diese Regeln können dabei alle in den expliziten Wissensressourcen vorhandenen Konzepte und Verbindungen verwenden und so auf der Basis des Dokumentengedächtnisses Wissen zur Disambiguierung formulieren. Während die regelbasierte Disambiguierung sehr flexibel ist, benötigt sie doch einen gewissen manuellen Aufwand. Die thematische Disambiguierung (vgl. Kapitel 7.4.6) dagegen kann vollkommen automatisch durchgeführt werden, indem einfach die thematische Überlappung der an der Ambiguität beteiligten Konzepte untersucht wird. So kann das in den expliziten Wissensressourcen kodierte Wissen ohne zusätzlichen

manuellen Aufwand direkt zur Ambiguitätsauflösung ausgenutzt werden.

In dieser Arbeit wurden die wichtigsten grundlegenden Aspekte der semantischen Indexierung mit expliziten Wissensressourcen dargestellt. Es wurden dabei einige weiterführende Themen ausgeklammert. So wurde in Hinblick auf Suchanfragen dargestellt, wie durch relationale Suchanfragen die ausgezeichneten Verbindungen insbesondere in Ontologien für gezielte Suchen ausgenutzt werden können. Darüber hinaus erscheint es naheliegend, die expliziten Wissensressourcen auch schon zur Identifizierung von Konzepten in den Suchanfragen zu verwenden. Die Relationen und Konzepte in Ontologien können so zu einer vereinfachten Referenzierung von Konzepten in Suchanfragen herangezogen werden.

So könnte zum Beispiel mit einer Suchanfrage $?\,\mathit{Birth}(\{\mathit{Philippe\ Pétain}\})$ (vgl. Kapitel 5.5) auf der Ontologie „Erster Weltkrieg" nach Dokumenten im semantischen Index gesucht werden, die das in der Ontologie ausgezeichnete Geburtsdatum des Konzepts *Philippe Pétain* enthalten. Benutzer müssen hierbei nicht die genauen Fakten der gewünschten Suchanfragen kennen und es ist ausreichend, den grundlegenden Aufbau der verwendeten Wissensressource zu kennen.

In dem oben genannten Beispiel bezeichne $\mathit{Birth}(\{\mathit{Philippe\ Pétain}\})$ die *Bildmenge* des Konzepts $\mathit{Philippe\ Pétain} \in \mathcal{W}_K$ unter der Relation $\mathit{Birth} \in \mathcal{W}_R$ in der Ontologie. Durch solche, hier nur kurz skizzierte Erweiterungen von Suchanfragen, kann das in den expliziten Wissensressourcen kodierte Weltwissen nicht nur für die Indexierung, sondern darüber hinaus auch für Suchanfragen ausgenutzt werden.

Eine weitaus wichtigere Fortsetzung dieser Arbeit stellt jedoch die Untersuchung der Frage dar, inwieweit das semantische Wissen, wie es in den expliziten Wissensressourcen der semantischen Indexierung kodiert ist, mit neueren Entwicklungen in der Computerlinguistik verbunden werden kann. Hierbei scheint eine besonders interessante Fragestellung zu sein, wie das strukturierte, zumeist manuell erstellte, Weltwissen in den expliziten Wissensressourcen dazu verwendet werden kann, verbreitete *Deep-Learning* Ansätze, wie insbesondere *word-embeddings* mit *word2vec* [40], mit eben diesem Wissen anzureichern. Dabei gilt es zu untersuchen, in wieweit sich die expliziten Wissensressourcen dazu eignen, solche maschinellen Lernverfahren noch weiter zu verbessern.

Andersherum erscheint es ebenso reizvoll zu untersuchen, wie word-embeddings zur Erweiterung expliziter Wissensressourcen verwendet werden könnten. Hierzu könnten unter anderem auch die verschiedenen Vektorähnlichkeiten unterschiedlicher Wörter und Phrasen von word-embeddings ausgenutzt werden, um bestehende Wissensressourcen zu verbessern und zu erweitern.

A. Installation der Anwendungssoftware

Die Anwendung zur semantischen Indexierung mit expliziten Wissensressourcen Semix ist in go[1] implementiert. Der Quellcode steht zur freien Verfügung[2]. Die Implementierung von Semix hängt von einer Anzahl weiterer Pakete ab. Hierunter auch einer Implementierung der minimierten Automaten mit konzeptuellen Zuordnungen[3].

Semix ist als monolithische, statisch gelinkte Anwendung implementiert und es werden zur Verwendung von Semix keine weiteren Ressourcen benötigt. Die unterschiedlichen Funktionalitäten der semantischen Indexierung sind in Semix über verschiedenen Unterkommandos realisiert (vgl. Appendix B.5).

A.1. Herunterladen der Binärdateien

Semix kann auch als vorkompilierte Binärdatei heruntergeladen werden. Unter der Download-Seite[4] von Semix können verschiedene Versionen von Semix für unterschiedliche Betriebssysteme[5] und Architekturen[6] heruntergeladen werden. Die Dateien für den Web-Server (vgl. Appendix B.5.7) können von dieser Seite aus ebenfalls heruntergeladen werden.

```
flo@nostromo:~$ wget http://bitbucket.org/fflo\
                /downloads/semix-linux-386
flo@nostromo:~$ wget http://bitbucket.org/fflo\
                /downloads/semix-http.tar.gz
flo@nostromo:~$ tar xf semix-html.tar.gz
flo@nostromo:~$ ./semix-linux-386 httpd html
```

[1] https://golang.org/
[2] https://bitbucket.org/fflo/semix
[3] https://bitbucket.org/fflo/sparsetable
[4] https://bitbucket.org/fflo/semix/downloads/
[5] Windows, Linux und OS X
[6] amd64/x86 64 und i386

A.2. Kompilierung von Semix

Um Semix kompilieren zu können, müssen auf dem System *go* und *git*[7] installiert sein und der Quellcode von Semix verfügbar sein. Im einfachsten Fall kann Semix installiert werden, indem folgender Befehl ausgeführt wird:

```
flo@nostromo:~$ go get -u bitbucket.org/fflo/semix
```

Der obige Befehl lädt den Quellcode nach `$GOPATH/src/...` herunter und installiert Semix nach `$GOPATH/bin`. Folgender Befehl lädt den Quellcode aus dem Repositorium herunter, wechselt in das Verzeichnis und kompiliert das Programm nach `semix`:

```
flo@nostromo:~$ mkdir -p $GOPATH/src/bitbucket.org\
                /fflo
flo@nostromo:~$ cd $GOPATH/src/bitbucket.org/fflo
flo@nostromo:~$ git clone bitbucket.org/fflo/semix
flo@nostromo:~$ cd semix
flo@nostromo:~$ go get -u ./...
flo@nostromo:~$ go build -o semix main.go
```

Eine weitere Möglichkeit besteht darin, eine bestimmte Version von Semix als Quellcodearchiv herunterzuladen[8], das Archiv zu entpacken und Semix zu kompilieren[9]:

```
flo@nostromo:~$ wget http://bitbucket.org/fflo\
                /semix/get/v1.2.1.tar.gz
flo@nostromo:~$ tar xf v1.2.1.tar.gz
flo@nostromo:~$ cd v1.2.1
flo@nostromo:~$ go get -u ./...
flo@nostromo:~$ go build -o semix main.go
```

[7] `https://git-scm.com/`

[8] `https://bitbucket.org/fflo/semix/downloads/?tab=tags`

[9] Auch in diesem Fall, müssen sowohl `git` als auch `go` auf dem System installiert sein, da `go get` implizit von `git` abhängt.

A.3. Konfiguration der Indexgröße

Semix kann mit verschiedenen Indexgrößen kompiliert werden (vgl. Kapitel 8.3.1 und 8.9.1). Hierzu können bei der manuellen Kompilierung von Semix (siehe oben) sogenannte *build tags*[10] angegeben werden, welche die Größe der von Semix verwendeten Indexeinträge angeben.

```
flo@nostromo:~$ go build -tags isizeN -o semix
```

Es stehen insgesamt fünf unterschiedliche Indexgrößen zur Auswahl:

- `isize1`: Textbelege werden *nicht* in den Index geschrieben.
- `isize2`: Textbelege und deren Positionen werden *nicht* in den Index geschrieben.
- `isize3`: Textbelege und die Relation indirekter Indexeinträge werden *nicht* in den Index geschrieben.
- `isize4`: Textbelege, Positionsangaben und die Relation indirekter Indexeinträge werden *nicht* in den Index geschrieben.
- `isize5`: Die Relation indirekter Indexeinträge werden *nicht* in den Index geschrieben.

Um Semix mit EFGT-Netzen zu verwenden und um die redundante Information der verwendeten einzigen Relation bei indirekten Indexeinträgen *nicht* zu speichern — wohl aber die Textbelege und deren Positionen — kann Semix folgendermaßen kompiliert werden:

```
flo@nostromo:~$ go build -tags isize5 -o semix
```

[10] `https://golang.org/pkg/go/build/`

B. Verwendung der Anwendungssoftware

B.1. Anwendungen

Semix ist als Client- und Serveranwendung implementiert. Das Programmpaket umfasst eine einzelne ausführbare Datei, über die alle Funktionalitäten der semantischen Indexierung mit expliziten Wissensressourcen implementiert sind.

Neben dem Aufbau der internen Wissensressourcen, der Indexierung von Dokumenten und Suchanfragen, stehen auch einige Hilfsfunktionalitäten zur Verfügung, mit denen unter anderem die Wissensressourcen durchsucht und Informationen über Konzepte abgefragt werden können. Darüber hinaus enthält die Anwendung einen einfachen Web-Server, mit dem die semantische Indexierung über ein einfaches Web-Frontend ausgeführt werden kann.

B.2. Konfiguration der Wissensressource

Die Konfiguration der Wissensressource, die zur semantischen Indexierung verwendet werden soll, geschieht über eine einfache Konfigurationsdatei. Mit den Informationen in der Konfigurationsdatei baut die Anwendung die internen Wissensressourcen auf.

```
[file]
path = "resource.xml"
type = "RDFXML"
cache = "/tmp/resource.xml.bin"
ambigs = "merge"

[predicates]
ignore = [
    "http://www.w3.org/2004/02/skos/core#narrower",
]
transitive = [
```

```
    "http://www.w3.org/2004/02/skos/core#broader",
    "http://www.w3.org/2004/02/skos/core#narrower",
]
name = [
    "http://www.w3.org/2004/02/skos/core#prefLabel",
]
distinct = [
    "http://www.w3.org/2004/02/skos/core#prefLabel",
    "http://www.w3.org/2004/02/skos/core#altLabel",
]
rules = [
    "http://rdf.internal.topiczoom.de/predicates/rule",
]
symmetric = []
inverted = []
ambiguous = []
```

Die TOML-basierte[1] Konfigurationsdatei besteht aus zwei Blöcken. Im `file`-Block werden die grundlegenden Parameter der Wissensressource festgelegt. Der `predicates`-Block dient der Konfiguration der Eigenschaften von unterschiedlichen Relationen in der Wissensressource.

- `file.path` definiert den Pfad der Eingabedatei der Ressource.
- `file.type` definiert das Eingabeformat der Ressource (Groß- und Kleinschreibung wird ignoriert):
  - `RDFXML` steht für RDF-XML kodierte Wissensressourcen.
  - `Turtle` steht für Turtle kodierte Wissensressourcen.
- `file.cache` definiert den Pfad der Cache-Datei der Ressource:
  - Nach dem Aufbau der expliziten Wissensressourcen werden diese in die Cache-Datei serialisiert.
  - Wenn eine Cache-Datei beim starten von Semix existiert, werden die Ressourcen aus dem Cache eingelesen, anstatt aus den Eingabedateien neu erzeugt zu werden.
  - Ist ein leerer Pfad angegeben wird der Caching-Mechanismus nicht verwendet und die expliziten Wissensressourcen werden neu aus den Eingabedateien aufgebaut.

[1] `https://github.com/toml-lang/toml#toml`

- `file.ambigs` definiert wie interne Ambiguitäten beim Aufbau der Wissensressource behandelt werden sollen (Groß- und Kleinschreibung wird ignoriert):
  - `discard` weist Semix an, ambige Lexikoneinträge nicht in das Lexikon zu übernehmen.
  - `split` übernimmt die Ambiguitäten in die Wissensressource indem ein entsprechendes ambiges Konzept erzeugt wird (vgl. Kapitel 7.2.2).
  - `merge` löst die Ambiguitäten durch die Einführung eines vagen Konzepts auf (vgl. Kapitel 7.2.2).
  - `fail` lässt das Programm bei der Behandlung von Ambiguitäten automatisch scheitern. So können Wissensressourcen auf ungewollte Ambiguitäten hin getestet werden.
  - Eine Schwellenwert $0 \leq t \leq 1$ weist Semix an, Ambiguitäten auf der Basis einer einfachen Heuristik aufzulösen (vgl. Kapitel 8.3.2):
    - Wenn die prozentuale Übereinstimmung der verbundenen Konzepte $\leq t$ ist, werden ambige Lexikoneinträge mittels `merge` behandelt,
    - andernfalls mittels `split`.

- `predicates.ignore` gibt eine Liste von URIs an, welche beim Aufbau der Wissensressource von Semix ignoriert werden sollen. Alle Tripel, die ein Prädikat enthalten, welches in dieser Liste aufgeführt ist, werden beim Aufbau der Wissensressourcen übersprungen.

- `predicates.transitive` gibt eine Liste der URIs von transitiven Prädikaten in der Wissensressource an. Für solche Relationen werden beim Aufbau die transitiven Hüllen gebildet.

- `predicates.symmetric` gibt eine Liste der URIs von symmetrischen Prädikaten in der Wissensressource an. Für solche Relationen werden beim Aufbau die symmetrischen Hüllen gebildet.

- `predicates.inverted` gibt eine Liste von URIs an, deren Verbindungen umgekehrt werden sollen ($\langle S, P, O\rangle \rightarrow \langle O, P, S\rangle$).

- `predicates.name` gibt eine Liste von URIs an, deren Prädikate Vorzugsnamen von Konzepten definieren. Die Vorzugsnamen werden nur zur besseren Lesbarkeit der Konzepte verwendet und *nicht* für Lexikoneinträge verwendet (vgl. Kapitel 8.3.1).

- `predicates.distinct` gibt eine Liste von URIs an, deren Prädikate eindeutige Lexikoneinträge für Konzepte definieren. Sollen die Vorzugsnamen ebenfalls in das Lexikon übernommen werden, müssen die entsprechenden Prädikatsnamen auch in diese Liste eingefügt werden.

- `predicates.ambig` gibt eine Liste von URIs an, deren Prädikate ambige Lexikoneinträge für Konzepte definieren.

- `predicates.rules` gibt eine Liste von URIs an, deren Prädikate zur Definition von Regeln zur regelbasierten Disambiguierung (vgl. Kapitel 7.4.5) verwendet werden.

B.3. Suchanfragen

Semix unterstützt Suchanfragen an einen semantischen Index. Es werden alle in dieser Arbeit beschriebenen Anfragen unterstützt. Darunter auch die Erweiterungen zu toleranten und fehlertoleranten Suchen. Die Syntax der Suchanfragen in Semix ist dabei stark von der in dieser Arbeit verwendeten Syntax von Suchanfragen inspiriert.

Alle Suchanfragen in Semix starten immer mit einem vorangestellten `?`. Konzepte und Relationen können dabei immer durch ihre Vorzugsnamen, URIs oder Lexikoneinträge referenziert werden. Diese müssen dabei nur dann in Anführungszeichen gesetzt werden, wenn sie Leerzeichen enthalten oder auch als Zahlen interpretiert werden könnten.

Im folgendem sei $\langle \mathcal{W}_K, \mathcal{W}_R, \mathcal{W}_{Lex}, \kappa \rangle$ eine Wissensressource im Sinne von Definition 1.5.1 und seien $A, B, C \in \mathcal{W}_K$ Konzepte der Wissensressource, $Q = \{A, B\} \subseteq \mathcal{W}_K$ eine Suchanfrage und $R, S, T \in \mathcal{W}_R$ Relationen. Für Suchanfragen spielt es keine Rolle, ob die verwendeten Wissensressourcen ein EFGT-Netz oder eine Ontologie sind. Es können immer nur die Konzepte und Relationen in Suchanfragen verwendet werden, die auch tatsächlich in den expliziten Wissensressourcen vorhanden sind.

Einfache Suchanfragen nach einer Konzeptmenge $?\,Q$ (vgl. Kapitel 5.5) entsprechen in Semix dem Ausdruck `?(A,B)`.

Beispiel B.3.1. Die Suchanfrage `?("Schlacht um Verdun")`, Dokumenten in denen das Konzept *Schlacht um Verdun* gefunden wurden. Das angefragte Konzept muss in diesem Fall in doppelte Anführungszeichen gesetzt werden, da es Leerzeichen enthält. In der Suchanfrage `?(Verdun)`, kann das Konzept *Verdun* auch ohne doppelte Anführungszeichen verwendet werden.

Soll nach mehr als einem Konzept gesucht werden, müssen die angefragten Konzepte durch Kommata voneinander getrennt werden. So liefert die Suchanfrage `?("Schlacht um Verdun", Verdun)` Textbelege zu den Konzepten *Schlacht um Verdun* und *Verdun* zurück.

Thematische Suchanfragen $?\,\mathcal{W}_K^{-1}(Q)$ (vgl. Kapitel 5.5.2) entsprechen in Semix dem Ausdruck `?(*(A,B))`. Soll eine Suchanfrage auf eine bestimmte Menge $\mathcal{C} = \{R, S\}$ eingeschränkt werden, müssen die Relationen in der Suchanfrage angegeben werden. Der Ausdruck `?(R,S(A,B))` in Semix entspricht dann $?\,\mathcal{C}(Q)$ (vgl. Kapitel 5.5.3).

Beispiel B.3.2. Die Suchanfrage `?(*("Personen und Persönlichkeiten"))` führt eine thematische Suche nach dem Konzept *Personen und Persönlichkeiten* aus, das nach allen Dokumenten sucht, in denen ein Konzept gefunden werden konnte, welches mit *Personen und Persönlichkeiten* über irgendeine Relation der Wissensressource verbunden ist (vgl. Kapitel 5.5.2).

Die Suche `?(Type(PERS))` dagegen, sucht nur nach Dokumenten, in denen Konzepte vorkommen, die über die Relation *Type* mit dem Konzept *PERS* verbunden sind. Es sollen also (neben den Vorkommen des Konzepts *PERS*) nur Vorkommen von Personen gesucht.

Tolerante und fehlertolerante Suchanfragen (vgl. Kapitel 7.3.3 und Kapitel 6.6) müssen in Semix mit einem `*` und oder einer Zahl $k > 0$ markiert werden. So entspricht die tolerante Suche $?^*\,Q$ in Semix `?*(A,B)`. Eine fehlertolerante Suche $?_k\,Q$ entspricht in Semix `?k(A,B)`. Tolerante und fehlertolerante Suchen können auch kombiniert werden und $?_k^*\,Q$ entspricht in Semix entweder `?*k(A,B)` oder `?k*(A,B)`.

Der spezielle Operator `!` kann in Suchanfragen dazu verwendet werden, um für thematische Suchen bestimmte Relationen zu verbieten. Somit entspricht die Suchanfrage `?(!R,S(A,B))` in Semix einer Suchanfrage $?\,\mathcal{W}_R \setminus \{R, S\}(Q)$.

Beispiel B.3.3. Die Suchanfrage `?*(Computervirus)` sucht nach Vorkommen des Konzepts *Computervirus*, wobei auch ambige Textbelege — unter anderem zu den Lexikoneinträgen *„Virus"* und *„Viren"* — zugelassen sind.

`?3(*("Personen und Persönlichkeiten"))` sucht thematisch nach dem Konzept *Personen und Persönlichkeiten*, wobei auch Textbelege mit einem Fehler $l \leq 3$ zugelassen sind.

Will man thematisch (auf der Ontologie „Erster Weltkrieg") nach einem Datum suchen und soll dabei sicher gestellt werden, dass *keine* Textbelege zu Ereignissen an diesem Datum gefunden werden, kann mit Semix die Suchanfrage

`?(!Date("1916"))`[2] verwendet werden. So werden alle Textbelege zu Konzepten, die mit der *Date* Relation mit dem Datum verbunden sind aus der Ergebnismenge entfernt, nicht aber Textbelege zu Konzepten, die über die *Death* oder *Birth* Relation mit diesem Datum verbunden sind.

Es folgt die Syntax der Suchanfragen in Backus-Naur-Form. Die offensichtlichen Regeln für die beiden Nichtterminalsymbole ⟨*string*⟩ und ⟨*num*⟩ fehlen aus Gründen der Übersichtlichkeit. Das Nichtterminalsymbol ⟨*string*⟩ bezeichnet Zeichenketten in doppelten Anführungszeichen oder Zeichenketten ohne Leerzeichen, das Nichtterminalsymbol ⟨*num*⟩ (positiven) ganzzahligen Ausdrücke.

⟨*query*⟩ ::= '?' ⟨*query-opt*⟩ '(' ⟨*query-expr*⟩ ')'

⟨*query-opt*⟩ ::= ⟨*num*⟩ '*' | '*' ⟨*num*⟩ | '*' | ⟨*num*⟩ | ε

⟨*query-expr*⟩ ::= ⟨*constraint-expr*⟩ '(' ⟨*list-expr*⟩ ')' | ⟨*list-expr*⟩

⟨*constraint-expr*⟩ ::= '*' | ⟨*list-expr*⟩ | '!' ⟨*list-expr*⟩

⟨*list-expr*⟩ ::= ⟨*string*⟩ ⟨*rest*⟩

⟨*rest*⟩ ::= ',' ⟨*string*⟩ ⟨*rest*⟩ | ε

B.4. Disambiguierungsregeln

Die Disambiguierungsregeln, die von der regelbasierten Disambiguierung (vgl. Kapitel 7.4.5) verwendet werden, werden von Semix in eine interne Repräsentation überführt (vgl. Kapitel 8.8). Die Regeln können einzelnen Konzepten der Wissensressourcen über spezielle Prädikate zugeordnet werden (vgl. Appendix B.2). Sie folgen der Syntax einfacher arithmetischer Ausdrücke in Infixnotation.

Die Syntax unterstützt Gleitkommazahlen, welche intern als 64 Bit breite Gleitkommazahlen repräsentiert werden. Darüber hinaus werden Zeichenketten bzw. *Strings*, Boole'sche Ausdrücke und Mengen aus Strings, Zahlen und Boole'schen Variablen unterstützt. Strings müssen dabei immer URIs, Vorzugsnamen oder Lexikoneinträge von Konzepten der expliziten Wissensressourcen referenzieren.

[2] In diesem Fall muss das Datum in doppelten Anführungszeichen angegeben werden, da es ansonsten als Zahl und nicht als Konzept erkannt werden würde.

Es werden die üblichen arithmetischen Operatoren +, -, * und / auf Zahlen unterstützt. Ebenso sind die üblichen Vergleichsoperatoren =, < und > implementiert. Für Bool'sche Variablen wird das logische Oder (+), das logische Und (*) und die Negation (Präfix -) unterstützt. Darüber hinaus können auch verschiedene Funktionen, wie zum Beispiel `exp(x)` zur Berechnung von $\exp x$ bzw. e^x verwendet werden.

Ebenso ist es möglich auf das aktuelle Dokumentengedächtnis zuzugreifen. So entspricht die Funktion `cs("A")` der Anzahl der Vorkommen des Konzepts A im Gedächtnis $count(\mathcal{M}_d^N, A)$, wobei die Funktion implizit auf das aktuelle Dokumentengedächtnis zugreift. Alle Operatoren weisen die übliche Assoziativität und Präzedenz auf. Ausdrücke können mit `(,)` geklammert werden.

Beispiel B.4.1. Die Regeln unterstützen den $\leq$ Vergleichsoperator nicht. Der Ausdruck $x \leq y$ kann mit den Regeln unter anderem auch als `-(x>y)` oder `(x=y)+(x<y)` ausgedrückt werden. Dies entspricht dem Ausdruck $\neg(x > y)$ bzw. $(x = y) \vee (x < y)$.

B.4.1. Operationen und Funktionen

Die Regeln unterscheiden zwischen Typen. Es gibt insgesamt 3 Typen:

1. `b` für Bool'sche Variablen `true` oder `false`
2. `n` für Gleitkommazahlen
3. `s` für Strings in doppelten Anführungszeichen

Jeder Typ kann zu einer Menge zusammengefasst werden. Dabei ist es nicht möglich Mengen zu verschachteln. Es ergeben sich somit genau 3 weitere Typen von Mengen:

1. `{b}` für Mengen aus Bool'schen Variablen
2. `{n}` für Menge aus Zahlen
3. `{s}` für Mengen von Strings

Alle binären Operatoren und Funktionen arbeiten immer nur auf einem Typ. Wenn die Typen der Operation nicht gleich sind oder von einem Operator oder einer Funktion nicht unterstützt werden, wird beim Aufbau der Wissensressourcen ein entsprechender Fehler ausgegeben und der Aufbau der expliziten Wissensressourcen scheitert. Ebenso werden keine Typkonvertierungen unterstützt. So können unter anderem auch Zahlen nicht in den Typ `b` konvertiert werden

oder umgekehrt. Einige der Operatoren und Funktionen sind dagegen *überladen* und arbeiten auf verschiedenen Typen. Einigen der Funktionen arbeiten auf einer variablen Anzahl von Typen.

Der folgende Überblick stellt alle verfügbaren Operatoren und Funktionen dar. Es sind dabei immer die Typen der Parameter und der Rückgabetyp angegeben. Dabei steht `T` für einen beliebigen Typen und *{T}* für eine Menge beliebiger (gleicher) Typen. `T...` bezeichnet immer eine variable Anzahl von Parametern. Da Mengen grundsätzlich nicht geschachtelt werden können (siehe oben), können variable Parameter entweder als Mengen (`x({a,b,c})` oder als Liste von Parametern (`x(a,b,c)`) spezifiziert werden.

- Operator +
  - `b+b -> b`: logisches Oder
  - `n+n -> n`: Addition zweier Zahlen
  - `{T}+{T} -> {T}`: Durchschnitt zweier Mengen
- Operator –
  - `n-n -> n`: Subtraktion zweier Zahlen
  - `{T}-{T} -> {T}`: Differenz zweier Mengen
  - `-b -> b`: Bool'sche Negation
  - `-n -> n`: Negation
- Operatoren * und /
  - `b*b -> b`: logisches Und
  - `n*n -> n`: Multiplikation zweier Zahlen
  - `{T}*{T} -> {T}`: Vereinigung zweier Mengen
  - `n/n -> n`: Division zweier Zahlen
- Operator =
  - `T=T -> b`: Äquivalenz
  - `{T}={T} ->b`: Mengenäquivalenz
  - `b<b -> b`: Vergleich kleiner-als (`false<true`)
  - `n<n -> b`: Vergleich kleiner-als
  - `b>b -> b`: Vergleich größer-als (`true>false`)
  - `n>n -> b`: Vergleich größer-als

- Funktionen `pow`, `log` und `exp`
  - `pow(n,n) -> n`: Berechnet x^y
  - `log(n) -> n`: Natürlicher Logarithmus (allgemeiner Logarithmus $\log_x y$: `log(x)/log(y)`)
  - `exp(n) -> n`: Berechnet e^x
- Funktionen `len`, `min` und `max`
  - `len(T…) -> n`: Gibt die Länge der Liste bzw. Menge zurück
  - `min(T…) -> T`: Gibt das minimale Element einer Liste bzw. Menge zurück
  - `max(T…) -> T`: Gibt das maximale Element einer Liste bzw. Menge zurück
- Funktionen `c` und `cs`
  - `c(s…) -> {n}`: Berechnet $\{count(\mathcal{M}_d^N, K) | K \in X\}$
  - `cs(s…) -> {n}`: Berechnet $\{count^*(\mathcal{M}_d^N, K) | K \in X\}$
- Funktionen `n`, `e` und `es`
  - `n() -> n`: Berechnet $len(\mathcal{M}_d^N)$
  - `e() -> {s}`: Berechnet $elems(\mathcal{M}_d^N)$
  - `es() -> {s}`: Berechnet $elems^*(\mathcal{M}_d^N)$

B.4.2. Verwendung der Regeln

Regeln müssen immer den Typen `b` zurückgeben. Die leere Regel ε gibt immer `false` zurück. Somit geben Konzepte, für die keine Regeln in der Wissensressource definiert sind, automatisch `false` zurück. Alle Regeln werden beim Aufbau der expliziten Wissensressourcen analysiert und in eine interne Repräsentation überführt. Die interne Repräsentation basiert auf einer Stack-Maschine, die mit Hilfe eines Stacks die Regeln berechnet und das Ergebnis zurückgibt.

Allen Regeln bei der regelbasierten Disambiguierung wird immer auch das aktuelle Dokumentengedächtnis übergeben. Funktionen, die auf das Dokumentengedächtnis zugreifen, können so immer implizit auf das Gedächtnis zugreifen. Die Beiden Funktionen `c` und `cs`, die den Funktionen $count(\mathcal{M}_d^N, K)$ bzw. $count^*(\mathcal{M}_d^N, K)$ entsprechen, akzeptieren im Gegensatz zur Darstellung in Kapitel 7.4.3 eine Menge von Konzepten und liefern eine Menge von Zahlen zurück. Dies dient vor allem der Verkettung mit den Funktionen `e` und `es`.

Beispiel B.4.2. Die beiden Regeln zur Disambiguierung von *Frankfurt(Main)* und *Frankfurt(Oder)* aus Beispiel 7.4.4 würden in Semix auf folgende Weise ausgedrückt werden:

- `cs("Frankfurt(Main)") > cs("Frankfurt(Oder)")`
- `cs("Frankfurt(Main)") < cs("Frankfurt(Oder)")`

Die beiden Regeln aus Beispiel 7.4.5 können in Semix auf folgende Weise ausgedrückt werden[3]:

- `(cs("A") / (max(cs(es()))))) > wk`
- `(cs("A") / (max(cs(es() - {"Topnode"})))) > wk`

B.4.3. Syntax

Es folgt die Syntax der Disambiguierungsregeln in Backus-Naur-Form. Die Regeln für die beiden Nichtterminalsymbole ⟨*string*⟩ und ⟨*num*⟩ fehlen aus Gründen der Übersichtlichkeit. Das Nichtterminalsymbol ⟨*string*⟩ bezeichnet Zeichenketten in doppelten Anführungszeichen, das Nichtterminalsymbol ⟨*num*⟩ beliebige (positive) Gleitkommazahlen.

⟨*rule*⟩ ::= ⟨*expr*⟩

⟨*expr*⟩ ::= ⟨*praefix-op*⟩ ⟨*expr*⟩
| ⟨*expr*⟩ ⟨*infix-op*⟩ ⟨*expr*⟩
| ‘(’ ⟨*expr*⟩ ‘)’
| ⟨*bool*⟩
| ⟨*num*⟩
| ⟨*str*⟩
| ⟨*set-expr*⟩
| ⟨*func-expr*⟩
| ⟨*expr*⟩
| ε

⟨*set-expr*⟩ ::= ‘{’ ⟨*bool-list*⟩ ‘}’ | ‘{’ ⟨*num-list*⟩ ‘}’ | ‘{’ ⟨*str-list*⟩ ‘}’

[3] Hierbei steht `wk` für einen konstanten Zahlausdruck, da die Syntax der Disambiguierungsregeln von Semix keine Variablen zulässt.

⟨*str-list*⟩ ::= ⟨*str*⟩ ⟨*str-rest*⟩ | ε

⟨*str-rest*⟩ ::= ‘`,`’ ⟨*str*⟩ ⟨*str-rest*⟩ | ε

⟨*bool-list*⟩ ::= ⟨*bool*⟩ ⟨*bool-rest*⟩ | ε

⟨*bool-rest*⟩ ::= ‘`,`’ ⟨*bool*⟩ ⟨*bool-rest*⟩ | ε

⟨*num-list*⟩ ::= ⟨*num*⟩ ⟨*num-rest*⟩ | ε

⟨*num-rest*⟩ ::= ‘`,`’ ⟨*num*⟩ ⟨*num-rest*⟩ | ε

⟨*func-expr*⟩ ::= ⟨*func-name*⟩ ‘`(`’ ⟨*func-args*⟩ ‘`)`’

⟨*func-args*⟩ ::= ⟨*expr*⟩ ⟨*func-rest*⟩ | ε

⟨*func-rest*⟩ ::= ‘`,`’ ⟨*expr*⟩ ⟨*func-rest*⟩ | ε

⟨*bool*⟩ ::= ‘`true`’ | ‘`false`’

⟨*infix-op*⟩ ::= ‘`-`’ | ‘`+`’ | ‘`/`’ | ‘`*`’ | ‘`<`’ | ‘`>`’ | ‘`=`’

⟨*praefix-op*⟩ ::= ‘`-`’

⟨*func-name*⟩ ::= ‘`exp`’
| ‘`log`’
| ‘`pow`’
| ‘`min`’
| ‘`max`’
| ‘`len`’
| ‘`e`’
| ‘`es`’

| ‘c’
| ‘cs’
| ‘n’

B.5. Verwendung von Semix

Semix ist eine monolithische Anwendung (vgl. Appendix A). Die verschiedenen Funktionen sind dabei als Unterkommandos realisiert.

In der Serveranwendung `daemon` ist die gesamte in dieser Arbeit dargestellten semantische Indexierung mit expliziten Wissensressourcen implementiert. Über das Unterkommando `put` können Dateien semantisch Indexiert werden. Mit dem Unterkommando `get` kann der semantische Index durchsucht werden. Alle weiteren Unterkommandos geben einen Überblick über die expliziten Wissensressourcen, die zur semantischen Indexierung verwendet werden. Sie ermöglichen es, verschiedene Informationen über die expliziten Wissensressourcen abzufragen.

Um einen Überblick über vorhandenen Unterkommandos und Kommandozeilenoptionen zu bekommen, kann folgender Aufruf verwendet werden[4]:

Beispiel B.5.1. Allgemeine Informationen zur Verwendung von Semix:

```
flo@nostromo:~$ semix --help
```

Um Informationen über ein Unterkommando zu erhalten, kann die `--help` Option auf dem entsprechenden Unterkommando ausgeführt werden:

```
flo@nostromo:~$ semix subcommand --help
```

Semix verfügt über eine Anzahl globaler Kommandozeilenparameter, die von allen Unterkommandos von Semix akzeptiert werden:

- `--help|-h` Gibt Hilfe zu Semix und alle weiteren Unterkommandos aus.
- `--json|-J` Schaltet JSON-formatierte Komandozeilenausgabe ein.
- `--verbose|-V` Schaltet ausführlichere Log-Ausgaben ein.
- `--daemon|-D` Setzt die `Host-Adresse:Port` des Servers. Falls ein benutzerdefinierter Host und oder Port angegeben werden, müssen für alle Kommandos die entsprechenden Host und Port Parameter angeben werden. Werden die Kommandozeilenparameter weggelassen, werden von allen Kommandos entsprechende Standardwerte verwendet.

[4] Alle folgenden Kommandozeilenbefehle setzen voraus, dass Semix entsprechend Appendix A.2 installiert wurde und die ausführbare Datei im Pfad gefunden werden kann.

B.5.1. Semix daemon

Semix ist als Client- und Serveranwendung konzipiert. Die Serveranwendung wird gestartet und interagiert mit den anderen Unterkommandos. Die Server-Anwendung erwartet eine Konfigurationsdatei (vgl. Kapitel B.2), mit deren Hilfe die interne Repräsentation der expliziten Wissensressourcen aufgebaut wird. Zum starten des Servers muss immer eine solche Konfigurationsdatei angegeben werden.

Kommandozeilenoptionen:

- `--index-size n` Setzt die Größe der Indexbuffer auf `n` (vgl. Kapitel 5.4.2).
- `--no-cache` Setzt das Lesen und Schreiben der expliziten Wissensressourcen in den Cache außer Kraft.
- `--dir|-d d` Setzt das Indexverzeichnis auf `d`. Standardmäßig wird entweder die Umgebungsvariable `$SEMIXPATH` oder, falls `$SEMIXPATH` nicht gesetzt ist, `$HOME/semix` verwendet.

Beispiel B.5.2. Folgender Befehl startet den Server mit einem Standard-Host und Standard-Port. Der Server startet im Vordergrund und andere Kommandos müssen von einem anderen Terminal aus gestartet werden.

```
flo@nostromo:~$ semix daemon kb.toml
```

Soll ein benutzerdefinierter Host-Parameter verwendet werden, müssen alle weiteren Unterkommandos ebenfalls diesen Parameter verwenden:

```
flo@nostromo:~$ semix daemon -D myhost:8888 kb.toml
# Auf einer anderen Konsole
flo@nostromo:~$ semix get -D myhost:8888 ...
```

B.5.2. Semix put

Das `put` Unterkommando kann dazu verwendet werden, um unterschiedliche Dateien semantisch zu indexieren. Dateien werden dabei auf den Server hochgeladen und dort semantisch indexiert. URLs werden vom Server eigenständig heruntergeladen und semantisch indexiert. Ist ein Verzeichnis angegeben, wird das Verzeichnis rekursiv durchlaufen und alle enthaltenen Dateien in den Index geschrieben. Alle indexierten Textbelege der Indexierung werden auf den Standarausgabekanal ausgegeben.

Über eine Reihe von Kommandozeilenoptionen können dem einfachen Indexierungsprozess verschiedene Disambiguierungsverfahren (vgl. Kapitel 7.4.1) und auch fehlertolerante Suchen hinzugefügt werden.

Kommandozeilenoptionen:

- `--local|-l` Gibt an, dass Dateien nicht auf den Server hochgeladen werden sollen. Der Server verwendet anstatt dessen die lokalen Pfade um die Dateien zu indexieren.
- `--resolver|-r d` Weist den Server an, bei der Indexierung eine der drei Disambiguierungen `d` zu verwenden. Dabei kann `d` für `thematic`, `ruled` und `simple` stehen. Um jeweils eine thematische (vgl. Kapitel 7.4.6), eine regelbasierte (vgl. Kapitel 7.4.5) oder eine einfache Disambiguierung anzuwenden. Dabei können auch mehrere Disambiguierungen angegeben werden. Die Disambiguierungen werden immer in der angegebenen Reihenfolge durchgeführt.
- `--levenshtein|-k n` Weist den Server an, bei der Indexierung eine fehlertolerante Suche mit einer Fehlerschranke von `n` zu verwenden (vgl. Kapitel 6.2). Es können hierbei mehrere Suchen mit verschiedenen Fehlerschranken angegeben werden.
- `--memory-size|-m n` Gibt die Größe des verwendeten Dokumentenkontexts an (vgl. Kapitel 7.4.3).
- `--threshold|-t n` Gibt den Schwellwert an, der bei der thematischen Disambiguierung verwendet werden soll (vgl. Kapitel 7.4.6),

Beispiel B.5.3. Sollen etwa alle Dateien in einem Verzeichnis `dir` indexiert werden, die einfache Disambiguierung angewendet werden und soll eine unscharfe Suchen mit den Fehlerschranken 1 und 3 ausgeführt werden, kann folgender Aufruf ausgeführt werden:

```
flo@nostromo:~$ semix put -k 1 -k 3 -r simple dir
```

B.5.3. Semix get

Mit dem `get` Unterkommando können Suchanfragen an den semantischen Index gestellt werden. Das Kommando schickt die Suchanfragen an den Server und schreibt die Ergebnisse in den Standarausgabekanal.

Die Syntax der Suchanfragen folgt dabei der in Appendix B.3 dargestellten Grammatik. Die Konzepte und Relationen können über ihre URIs, ihre Vorzugsnamen oder deren Lexikoneinträge (im Falle von Konzepten) referenziert werden.

Kommandozeilenoptionen:

- `--skip|-s n` Weist den Server an, die ersten `n` Ergebnisse der Suchanfrage zu überspringen.

- `--max|-m n` Weist den Server an, maximal `n` Ergebnisse der Suchanfrage auszugeben.

Beispiel B.5.4. Eine thematische Suche nach dem Konzept *Gerhard Schröder* kann folgendermaßen ausgeführt werden:

```
flo@nostromo:~$ semix get '?(*("Gerhard␣Schröder"))'
```

B.5.4. Semix version

Das Unterkommando `version` gibt einfach nur die aktuelle Version der verwendeten Anwendung aus.

B.5.5. Semix search

Das `search` Unterkommando sucht nach Konzepten und Relationen in den expliziten Wissensressourcen und gibt die Ergebnisse auf dem Standarausgabekanal aus. Konzepte und Relationen können dabei — wie auch bei Suchanfragen mit `get` — entweder über ihre URIs, Vorzugsnamen oder Lexikoneinträge gesucht werden.

Kommandozeilenoptionen:

- `--predicate|-p` Weist den Server an, nicht nach Konzepten sondern nach Prädikaten bzw. Relationen zu suchen

B.5.6. Semix info

Das Unterkommando `info` gibt Informationen über ein Konzept oder eine Relation auf dem Standarausgabekanal aus. Es werden sämtliche verfügbaren Informationen — darunter auch alle verbunden Konzepte und alle Lexikoneinträge — ausgegeben. Die Angefragten Konzepte oder Relationen müssen eindeutig mit ihrer URI oder internen Identifikationsnummer angesprochen werden.

Dieses Unterkommando dient dazu, die Informationen über ein eindeutig identifiziertes Konzept oder Prädikat auszugeben. Da es die URI oder Identifikationsnummer des angefragten Konzepts benötigt, kann es auch in Kombination mit dem `search` Unterkommando verwendet werden um Informationen über bestimmte Konzepte oder Prädikate der expliziten Wissensressourcen abzufragen.

Beispiel B.5.5. So kann unter anderem `jq`[5] zur Weiterverarbeitung von JSON verwendet werden, um Informationen über die Konzepte oder Prädikate der expliziten Wissensressourcen anzufragen, indem Konzepte mit `semix search` gesucht werden und an `semix info` weitergeleitet werden:

[5] `https://stedolan.github.io/jq/`

```
flo@nostromo:~$ semix info -V \
    $(semix search -J Virus | jq -r'.[0].ID')
2018/02/22 13:35:22 [DBUG] sending request \
    [GET] http://localhost:8888/info?id=63
63:1:1: Virus (63)
63:1:1: + 0 a-star (64)->Computerviren (17)
63:1:1: + 0 a-star (64)->Viren (Mikroorganismen) (40)
63:1:1: - Virus
63:1:1: - Viren
```

B.5.7. Semix httpd

In Semix ist auch ein einfacher Web-Server implementiert, mit dem die semantische Indexierung über ein einfaches Web-Frontend ausgeführt werden kann. Hierzu muss das Unterkommando `httpd` verwendet werden, um einen HTTP-Server zu starten, der dann mit einem Web-Browser angesprochen werden kann.

Wie alle anderen Unterkommandos in Semix, benötigt auch der Web-Server eine laufende Server-Instanz der semantischen Indexierung. Der Web-Server startet standardmäßig im Vordergrund und lauscht auf Anfragen nach *localhost* auf Port 8080. Ist der Web-Server gestartet, kann man das Web-Frontend mit einem Webbrowser über die Adresse `http://localhost:8080` erreichen.

Der Web-Server bietet eine einfache Schnittstelle, um die semantische Indexierung von Dateien und URLs durchzuführen. Dabei können ebenso verschiedene fehlertolerante Suchen und Disambiguierungsalgorithmen für die Indexierung interaktiv eingestellt werden. Darüber hinaus können auch die verwendeten Wissensressourcen untersucht werden. So können unter anderem auch die Konzepte in den Wissensressourcen, deren Verbindungen und Lexikoneinträge analysiert werden (vgl. Abbildung B.1).

Zum Starten des HTTP-Servers wird ein Verzeichnis benötigt, in welchem dessen HTML Dateien liegen (vgl. Appendix A.1). Sie müssen dem Unterkommando als Argument mitgeliefert werden.

Kommandozeilenoptionen:

- `--host|-H` Setzt die Adresse des Web-Servers.

Beispiel B.5.6. Folgender Aufruf starten den Web-Server, der auf Localhost auf Port 80 lauscht und dessen HTML-Daten im `/srv/html` Verzeichnis liegen:

```
flo@nostromo:~$ semix daemon -H [::1]:80 /srv/html
```

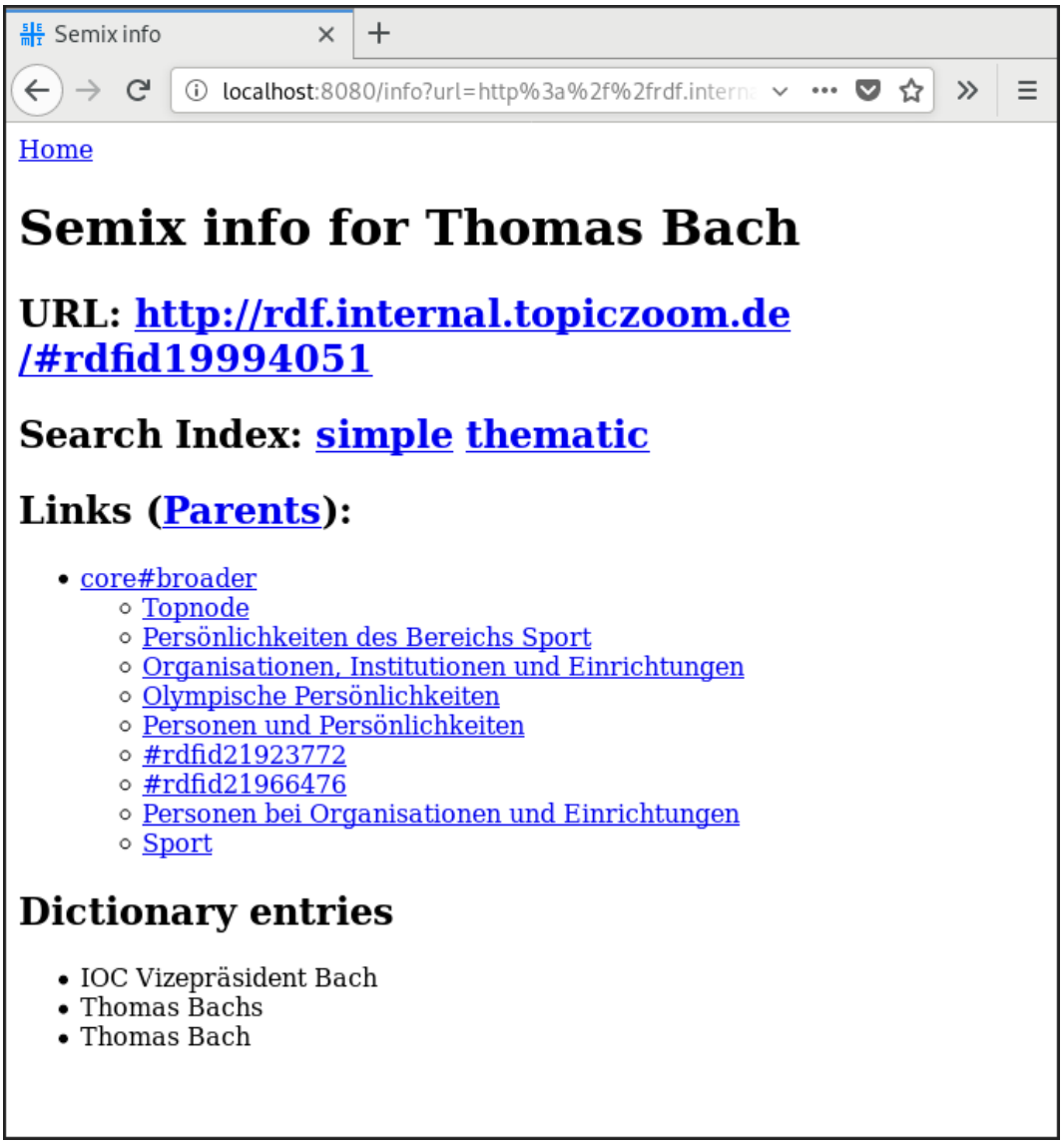


Abbildung B.1.: Screenshot des Web-Servers. Mit dem Web-Server können nicht nur Dateien indexiert und Suchanfragen gestellt werden, sondern es können auch die verwendeten Wissensressourcen interaktiv durchlaufen werden.

Literatur

[1] USA World Wide Web Consortium (W3C). *RDF Primer*. https://www.w3.org/TR/rdf-primer. 2004.

[2] USA World Wide Web Consortium (W3C). *RDQL - A Query Language for RDF*. http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/. 2004.

[3] USA World Wide Web Consortium (W3C). *SPARQL Recomendation 1.1*. http://www.w3.org/TR/sparql11-overview/. 2013.

[4] USA World Wide Web Consortium (W3C). *Turtle - Terse RDF Triple Language*. https://www.w3.org/TeamSubmission/turtle. 2011.

[5] Alfred V. Aho und Margaret J. Corasick. „Efficient String Matching: An Aid to Bibliographic Search“. In: *Commun. ACM* 18.6 (Juni 1975), S. 333–340. ISSN: 0001-0782. DOI: 10.1145/360825.360855. URL: http://doi.acm.org/10.1145/360825.360855.

[6] *Bash Reference Manual*. English. Version Version 4.4. GNU. URL: https://www.gnu.org/software/bash/manual/html_node/.

[7] Dave Beckett. *Redland librdf RDF API Library*. 2005. URL: http://librdf.org.

[8] Thomas Breul. „The hOCR Embedded OCR Workflow and Output Format“. In: (März 2010).

[9] Levin Brunner, Klaus U. Schulz und Felix Weigel. „Organizing Thematic, Geographic and Temporal Knowledge in a Well-founded Navigation Space: Logical and Algorithmic Foundations for EFGT Nets“. In: *International Journal of Web Services Research* (2005).

[10] T. Levin Brunner. *Aufbau und Verwendung großer EFGT-Netze*. Martin Meidenbauer Verlag, 2008. ISBN: 9783899756937.

[11] Jan Daciuk u. a. „Incremental Construction of Minimal Acyclic Finite-State Automata“. In: *Computational Linguistics* 26 (2000).

[12] Mark Davis. *Unicode nearing 50% of the web*. 28. Jan. 2010. URL: https://googleblog.blogspot.de/2010/01/unicode-nearing-50-of-web.html.

[13] Scott Deerwester u. a. „Indexing by Latent Semantic Analysis“. In: *Journal of the American society for information science* (1990).

[14] E. Ukkonen. „Algorithms for Approximate String Matching“. In: *Information and Control* 64 (1985), S. 100–118.

[15] Andrea Ernst-Gerlach und Norbert Fuhr. „Generating Search Term Variants for Text Collections with Historic Spellings“. In: *Advances in Information Retrieval: 28th European Conference on IR Research, ECIR 2006, London, UK, April 10-12, 2006. Proceedings*. Hrsg. von Mounia Lalmas u. a. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, S. 49–60. ISBN: 978-3-540-33348-7. DOI: 10.1007/11735106_6. URL: http://dx.doi.org/10.1007/11735106_6.

[16] Andrea Ernst-Gerlach und Norbert Fuhr. „Generating Search Term Variants for Text Collections with Historic Spellings“. In: *Advances in Information Retrieval*. Hrsg. von Mounia Lalmas u. a. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, S. 49–60. ISBN: 978-3-540-33348-7.

[17] Andrea Ernst-Gerlach und Norbert Fuhr. „Retrieval in Text Collections with Historic Spelling Using Linguistic and Spelling Variants“. In: *Proceedings of the 7th ACM/IEEE-CS Joint Conference on Digital Libraries*. JCDL '07. Vancouver, BC, Canada: ACM, 2007, S. 333–341. ISBN: 978-1-59593-644-8. DOI: 10.1145/1255175.1255242. URL: http://doi.acm.org/10.1145/1255175.1255242.

[18] Andrea Ernst-Gerlach und Norbert Fuhr. *Retrieval in Text Collections with Historic Spelling using Linguistic and Spelling Variants*. Jan. 2007.

[19] Florian Fink. „Flexible Dokumenten Indexierung“. Magisterarb. Ludwigs-Maximilians- Universität (LMU) München, 2011. URL: http://www.cis.lmu.de/~finkf/fdi.pdf.

[20] Florian Fink, Christoph Ringlstetter und Klaus Schulz. „Automated Assignment of Topics to OCRed Historical Texts“. In: *Proceedings of the First International Conference on Digital Access to Textual Cultural Heritage*. DATeCH '14. Madrid, Spain: ACM, 2014, S. 23–28. ISBN: 978-1-4503-2588-2. DOI: 10.1145/2595188.2595206. URL: http://doi.acm.org/10.1145/2595188.2595206.

[21] Florian Fink, Klaus-U. Schulz und Uwe Springmann. „Profiling of OCR'ed Historical Texts Revisited“. In: *ArXiv e-prints* (2017). URL: http://arxiv.org/abs/1701.05377.

[22] Michael R. Genesereth und Nils J. Nilsson. „{CHAPTER} 2 - Declarative Knowledge“. In: *Logical Foundations of Artificial Intelligence*. Hrsg. von Michael R. Genesereth und Nils J. Nilsson. San Francisco (CA): Morgan Kaufmann, 1987, S. 9–44. ISBN: 978-0-934613-31-6. DOI: \url{https://doi.org/10.1016/B978-0-934613-31-6.50008-2}. URL: %5Curl%7Bhttps://www.sciencedirect.com/science/article/pii/B9780934613316500082%7D.

[23] Richard Gillam. *Unicode Demystified: A Practical Programmer's Guide to the Encoding Standard*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN: 0201700522.

[24] Annette Gotscharek u. a. „Towards Information Retrieval on Historical Document Collections: The Role of Matching Procedures and Special Lexica“. In: *Int. J. Doc. Anal. Recognit.* 14.2 (Juni 2011), S. 159–171. ISSN: 1433-2833. DOI: 10.1007/s10032-010-0132-6. URL: http://dx.doi.org/10.1007/s10032-010-0132-6.

[25] W3C/IETF URI Planning Interest Group. *URIs, URLs, and URNs: Clarifications and Recommendations 1.0*. https://www.w3.org/TR/uri-clarification/. 2001.

[26] Thomas R. Gruber. „A translation approach to portable ontology specifications“. In: *Knowledge Acquisition* 5.2 (1993), S. 199–220. ISSN: 1042-8143. DOI: https://doi.org/10.1006/knac.1993.1008. URL: http://www.sciencedirect.com/science/article/pii/S1042814383710083.

[27] Thomas R. Gruber. „Toward principles for the design of ontologies used for knowledge sharing?“ In: *International Journal of Human-Computer Studies* 43.5 (1995), S. 907–928. ISSN: 1071-5819. DOI: https://doi.org/10.1006/ijhc.1995.1081. URL: %5Curl%7Bhttp://www.sciencedirect.com/science/article/pii/S1071581985710816%7D.

[28] Nicola Guarino, Daniel Oberle und Steffen Staab. „What Is an Ontology?“ In: *Handbook on Ontologies*. Hrsg. von Steffen Staab und Rudi Studer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, S. 1–17. ISBN: 978-3-540-92673-3. DOI: 10.1007/978-3-540-92673-3_0. URL: https://doi.org/10.1007/978-3-540-92673-3_0.

[29] Erik Hatcher, Otis Gospodnetic und Mike McCandless. *Lucene in Action*. 2nd revised edition. Manning, Aug. 2010. ISBN: 9781933988177. URL: http://amazon.de/o/ASIN/1933988177/.

[30] C. A. R. Hoare. „Communicating Sequential Processes“. In: *Commun. ACM* 21.8 (Aug. 1978), S. 666–677. ISSN: 0001-0782. DOI: 10.1145/359576.359585. URL: http://doi.acm.org/10.1145/359576.359585.

[31] P. Hoffman und F. Yergeau. *UTF-16, an encoding of ISO 10646*. RFC 2781. RFC Editor, Feb. 2000, S. 1–13. URL: https://www.ietf.org/rfc/rfc2781.txt.

[32] ISO. *International Organization for Standardization*. URL: http://iso.org.

[33] *JTC1/SC2/WG2 - ISO/IEC 10646 - UCS*. Standard 10646. Colombo: International Organization for Standardization, Okt. 2014. URL: http://std.dkuug.dk/JTC1/SC2/WG2/.

[34] Dongwon Lee und Wesley W. Chu. „Comparative Analysis of Six XML Schema Languages“. In: *ACM SIGMOD Record* (2000).

[35] V. Levenshtein. „Binary codes capable of correcting spurious insertions and deletions of ones“. In: *Problems of Information Transmission* 1 (1965), S. 8–17.

[36] A. Linke, M. Nussbaumer und P.R. Portmann. *Studienbuch Linguistik*. Reihe Germanistische Linguistik, 121. Niemeyer Max Verlag GmbH, 1991. ISBN: 9783484311213. URL: http://books.google.de/books?id=aHMbAAAAIAAJ.

[37] Julie Beth Lovins. „Development of a stemming algorithm“. In: *Mechanical Translation and Computational Linguistics* 11 (1968), S. 22–31.

[38] Inc Merriam-Webster. *Merriam-Webster's Manual for Writers and Editors*. Merriam-Webster, 1998. ISBN: 9780877796220. URL: https://books.google.de/books?id=7C6V9zRxSPkC.

[39] Soyan Mihov und Klaus U. Schulz. „Efficient Finite-State Techniques for Knowledge-based Text Processing“. März 2017.

[40] T. Mikolov u. a. „Efficient Estimation of Word Representations in Vector Space“. In: *ArXiv e-prints* (Jan. 2013). arXiv: 1301.3781 [cs.CL].

[41] Gonzalo Navarro und Mathieu Raffinot. *Flexible Pattern Matching in Strings: Practical On-Line Search Algorithms for Texts and Biological Sequences*. 1st. New York, NY, USA: Cambridge University Press, 2007. ISBN: 0521039932.

[42] Michael A. Olson, Keith Bostic und Margo Seltzer. *Berkeley DB*.

[43] Pascal Hitzler and Markus Krötsch and Sebastian Rudolph and York Sure. *Semantic Web*. 1. Springer, 2008. ISBN: 9783540339939.

[44] Susanne Peters. „Ontologie “Erster Weltkrieg““. Magisterarb. Ludwigs- Maximilians-Universität (LMU) München, 2011.

[45] M.F. Porter. „An algorithm for suffix stripping“. In: *Program* 14(3) (1980), S. 130–137.

[46] R.A. Wagner und M. Fisher. „The string-to-string correction problem“. In: *Journal of the ACM* (1974).

[47] Ulrich Reffle. „Algorithmen und Methoden zur dokumentenspezifischen Analyse historischer und OCR-erfasster Texte“. Diss. Fakultät für Sprach- und Literaturwissenschaften der Ludwig-Maximilians-Universität München, 2011.

[48] G. Salton, A. Wong und C. S. Yang. „A Vector Space Model for Automatic Indexing“. In: *Commun. ACM* 18.11 (Nov. 1975), S. 613–620. ISSN: 0001-0782. DOI: 10.1145/361219.361220. URL: http://doi.acm.org/10.1145/361219.361220.

[49] Helmut Schmid. *Probabilistic Part-of-Speech Tagging Using Decision Trees*. 1994.

[50] Klaus U. Schulz und Felix Weigel. „Systematics and architecture for a resource representing knowledge about named entities“. In: *Principles and Practice of Semantic Web Reasoning* (2003).

[51] Monika Schwarz und Jeannette Chur. *Semantik*. 5. Reihe Germanistische Linguistik, 121. narr studienbücher, 2007. ISBN: 9783823362968.

[52] Uwe Springmann und Anke Lüdeling. „OCR of historical printings with an application to building diachronic corpora: A case study using the RIDGES herbal corpus“. In: *Digital Humanities Quarterly* 11.2 (2017). URL: http://www.digitalhumanities.org/dhq/vol/11/2/000288/000288.html.

[53] Aneta Stojić und Nataša Košuta. „Zur Abgrenzung von Mehrwortverbindungen“. In: *Zagreber Germanistische Beiträge* 21 (2012), S. 358–373.

[54] Stoyan Mihov und Klaus U. Schulz. „Fast Approximate Search in Large Dictionaries“. In: *Computational Linguistics* 30.4 (Dez. 2004), S. 451–477.

[55] Bjarne Stroustrup. *The C++ Programming Language*. fourth. Addision-Wesley, 2013.

[56] Robert Endre Tarjan und Andrew Chi-Chih Yao. „Storing a Sparse Table“. In: *Communications of the ACM* (Nov. 1979).

[57] Joshua Tauberer. *What Is RDF*. http://www.xml.com/pub/a/2001/01/24/rdf.html. Juli 2006.

[58] TEI Consortium, eds. „TEI P5: Guidelines for Electronic Text Encoding and Interchange 1.0“. In: (März 2010).

[59] The Library of Congress. „Structure of ALTO Files“. In: (2016).

[60] USA World Wide Web Consortium – W3C. *XML Schema Part 0: Primer Second Edition*. http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/. 2004.

[61] USA World Wide Web Consortium – W3C. *XML Schema Part 1: Primer Second Edition*. http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/. 2004.

[62] USA World Wide Web Consortium – W3C. *XML Schema Part 2: Primer Second Edition*. http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/. 2004.

[63] Wikipedia. *Frankfurt am Main — Wikipedia, The Free Encyclopedia*. [Online; accessed 17-November-2017]. 2017. URL: %5Curl%7Bhttps://de.wikipedia.org/wiki/Frankfurt_am_Main%7D.

[64] Wikipedia. *München — Wikipedia, The Free Encyclopedia*. [Online; accessed 03-März-2018]. 2018. URL: %5Curl%7Bhttps://de.wikipedia.org/wiki/M%C3%BCnchen%7D.

[65] Wikipedia. *Relation (Begriffsklärung)*. [Online; accessed 09-Dezember-2015]. 2015. URL: https://de.wikipedia.org/wiki/Relation_(Begriffskl%C3%A4rung).

[66] I.H. Witten, A. Moffat und T.C. Bell. *Managing gigabytes: compressing and indexing documents and images*. The Morgan Kaufmann series in multimedia information and systems. Morgan Kaufmann Publishers, 1999. ISBN: 9781558605701.

[67] L. Wittgenstein. *Philosophische Untersuchungen*. Suhrkamp Taschenbuch Wissenschaft. Suhrkamp, 1977. ISBN: 9783518078037. URL: http://books.google.de/books?id=4q88AAAAYAAJ.

[68] Sun Wu und Udi Manber. „Fast Text Searching Allowing Errors“. In: *Communications of the ACM* 35.10 (1992), S. 83–91.

[69] F. Yergeau. *UTF-8, a transformation format of ISO 10646*. RFC 3629. RFC Editor, Nov. 2003, S. 1–13. URL: https://tools.ietf.org/html/rfc3629.

Danksagung

An dieser Stelle möchte ich mich bei einigen Personen bedanken, die mich im Verlauf dieser Arbeit begleitet und unterstützt haben.

An erster Stelle steht natürlich Klaus Schulz, der mich geduldig unterstützt und mit seiner pragmatischen Art angeleitet hat. Die enge Zusammenarbeit mit ihm und insbesondere auch mit Uwe Springmann hat mich inspiriert und wachsen lassen.

Ich danke auch meinen vielen anderen Kollegen am CIS mit denen ich zusammenarbeiten konnte. Max Hadersbeck für unsere "Rauchpausen" und Wittgenstein, Robert Zangenfeind für seine tiefen linguistischen Einblicke, Peggy Hobmaier für ihre Unterstützung in den Untiefen der Bürokratie, Thomas Schäfer dass er den ganzen Laden am laufen hält und Daniel Bruder für die ewigen Gespräche über das Programmieren.

Ich möchte hier auch noch meiner ganzen Familie, meinen Geschwistern, meinen Eltern und Schwiegereltern für ihre Unterstützung danken. Insbesondere möchte ich auch meiner Mutter danken, die dieses ominöse Fach "Computerlinguistik" an der LMU erst für mich entdeckt hat.

Einen ganz besonderen Dank schulde ich meiner Frau Anna für unsere wundervolle gemeinsame Zeit, unser gemeinsames Studium, die Filme, die Konzerte und den Dinosauriern.