Karlsruhe Institute of Technology

Automatisches Auflösen von Abkürzungen in Quelltext

Masterarbeit
von

Gilbert Groten

An der Fakultät für Informatik
Institut für Programmstrukturen
und Datenorganisation (IPD)

Erstgutachter:	Prof. Dr. Walter F. Tichy
Zweitgutachter:	Prof. Dr. Anne Koziolek
Betreuender Mitarbeiter:	M.Sc. Tobias Hey

Bearbeitungszeit: 24. Oktober 2020 – 23. April 2021

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Die Regeln zur Sicherung guter wissenschaftlicher Praxis im Karlsruher Institut für Technologie (KIT) habe ich befolgt.

Karlsruhe, 23. April 2021

(**Gilbert Groten**)

Publikationsgenehmigung

Melder der Publikation

Hildegard Sauer

Institut für Programmstrukturen und Datenorganisation (IPD)
Lehrstuhl für Programmiersysteme
Leiter Prof. Dr. Walter F. Tichy

+49 721 608-43934
hildegard.sauer@kit.edu

Erklärung des Verfassers

Ich räume dem Karlsruher Institut für Technologie (KIT) dauerhaft ein einfaches Nutzungsrecht für die Bereitstellung einer elektronischen Fassung meiner Publikation auf dem zentralen Dokumentenserver des KIT ein.

Ich bin Inhaber aller Rechte an dem Werk; Ansprüche Dritter sind davon nicht berührt.

Bei etwaigen Forderungen Dritter stelle ich das KIT frei.

Eventuelle Mitautoren sind mit diesen Regelungen einverstanden.

Der Betreuer der Arbeit ist mit der Veröffentlichung einverstanden.

Art der Abschlussarbeit:	Masterarbeit
Titel:	Automatisches Auflösen von Abkürzungen in Quelltext
Datum:	23. April 2021
Name:	Gilbert Groten

Karlsruhe, 23. April 2021

(**Gilbert Groten**)

Kurzfassung

Abgekürzte Quelltextbezeichner stellen ein Hindernis bei der Gewinnung von Informationen aus Quelltext dar. Im Rahmen dieser Arbeit werden Abkürzungsauflösungsverfahren entwickelt, um diese abgekürzten Quelltextbezeichner zu den gemeinten, nicht abgekürzten Begriffen aufzulösen. Zum einen wird die Entscheidung für den besten Auflösungskandidaten mittels worteinbettungsbasierten Ähnlichkeitsfunktionen getroffen. Zum anderen werden Trigramm-Grammatiken verwendet, um die Wahrscheinlichkeit eines Auflösungskandidaten zu bestimmen. Die im Rahmen dieser Arbeit entwickelten Verfahren bauen auf zwei Verfahren auf, welche von Alatawi et al. entwickelt wurden. In diesen werden statistische Eigenschaften von Quelltextabkürzungen, sowie Uni- und Bigramm-Grammatiken verwendet, um die Auflösung einer Abkürzung zu bestimmen. Das präziseste der im Rahmen dieser Arbeit entwickelten Verfahren (das Trigramm-basierte) löst auf einem Beispielquelltext, evaluiert gegen eine von Alatawi et al. bereitgestellte Musterlösung, 70,33% der abgekürzten Quelltextbezeichner richtig auf, und ist damit 3,30 Prozentpunkte besser als das nachimplementierte, präziseste Verfahren von Alatawi et al.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Bei der Analyse, Wartung und Erweiterung von Software muss einem Quelltext immer wieder die implementierte Information und Intention entnommen werden. Dies gilt sowohl für menschliche Entwickler, welche einen existierenden Quelltext lesen und gegebenenfalls verändern, als auch für automatische Verfahren der Informationsrückgewinnung. Für beides sind Quelltextbezeichner eine wichtige Grundlage. Sowohl ein menschlicher Leser, als auch ein auf Sprachverarbeitung basierendes automatisches Analysewerkzeug sind auf aussagekräftige Quelltextbezeichner angewiesen. Obwohl dies unter Entwicklern allgemein bekannt ist, werden häufig abgekürzte Quelltextbezeichner verwendet. Ist ein Leser nicht in der Lage den ursprünglichen Begriff, welcher in einem solchen Bezeichner abgekürzt wurde, zu erraten, oder hat ein automatisches Informationsrückgewinnungswerkzeug keinen oder keinen hinreichenden Mechanismus zur Auflösung solcher abgekürzten Bezeichner, hindert dies die Analyse der Software, und damit gegebenenfalls die Wartung und Weiterentwicklung. Um die durch die abgekürzten Bezeichner verschleierte Information zurückzugewinnen, wäre ein automatisches Werkzeug wünschenswert, welches Abkürzungen in Quelltext zu ausdrucksstarken Bezeichnern auflöst. Ein Verfahren zur automatischen Abkürzungsauflösung muss zunächst abgekürzte Bezeichner als solche erkennen. Außerdem muss es die Information, welche den aufzulösenden Bezeichnern fehlt, aus anderen Wissensquellen gewinnen. Quelltextbezeichner sind außerdem oftmals aus mehreren Token zusammengesetzt, welche nicht zwingend durch eindeutige Trennzeichen unterschieden werden können. Um eine Abkürzung, welche in einem solchen Bezeichner auftritt, auflösen zu können, muss diese aus dem Bezeichner isoliert werden. Zudem müssen gegebenenfalls Mehrdeutigkeiten aufgelöst werden, falls mehr als ein Kandidat als Auflösung einer Abkürzung in Frage kommt.

1.1 Zielsetzung

Ziel dieser Arbeit ist ein Verfahren zu entwickeln, welches automatisch Abkürzungen in Quelltext auflöst, um die vollständigen natürlichsprachlichen Begriffe, welche darin abgekürzt sind, zurückzugewinnen. Hierzu muss zunächst entschieden werden, welche Bezeichner aufgelöst werden sollen, also abgekürzte Bezeichner erkannt werden. Da Quelltextbezeichner oftmals aus mehreren Token zusammengesetzt sind, muss ein Bezeichner gegebenenfalls aufgespalten werden, damit die darin beinhalteten Abkürzungen aufgelöst werden können. Um die so gefundenen Abkürzungen aufzulösen, müssen Kandidaten ermittelt werden, welche als Auflösung in Frage kommen. Gibt es mehrere solcher Kandidaten,

muss unter diesen eine Entscheidung getroffen werden, um den Besten und somit hoffentlich die richtige Auflösung zu ermitteln. Letztendlich ist das Ziel des Verfahrens möglichst viele abgekürzte Bezeichner richtig zu dem Begriff aufzulösen, welche der Entwickler darin abgekürzt hat.

1.2 Aufbau der Arbeit

Diese Arbeit ist wie folgt strukturiert: In Kapitel 2 werden theoretische Grundlagen erörtert, welche für das Verständnis der folgenden Kapitel notwendig sind. Kapitel 3 beschreibt die Projektumgebung, welche für die Implementierung der hier vorgestellten Verfahren genutzt wird. In Kapitel 4 werden verwandte Arbeiten des gleichen Forschungsgebiets diskutiert, um den aktuellen Stand der Forschung zu erfassen und Inspiration für das folgende Kapitel 5 zu gewinnen, in welchem sowohl die zuvor beschriebenen verwandten Arbeiten diskutiert, als auch allgemeine Eigenschaften des Problems analysiert werden. Diese Analyse wird in Kapitel 6 genutzt, um neue Verfahren zur Abkürzungsauflösung zu entwerfen. Darauf folgt in Kapitel 8 eine Beschreibung der Datensätze, welche für die Umsetzung und Evaluation der Verfahren genutzt, beziehungsweise erzeugt wurden. Details zur Implementierung der vorgeschlagenen Verfahren werden in Kapitel 7 beschrieben. In Kapitel 9 werden die Ergebnisse dieser Arbeit evaluiert. Zuletzt enthält Kapitel 10 eine Zusammenfassung der Arbeit und einen Ausblick auf die zukünftige Forschung in diesem Gebiet.

2 Grundlagen

In diesem Kapitel werden die Grundlagen erörtert, welche für das Verständnis der Arbeit Voraussetzung sind. Dies beinhaltet insbesondere allgemeine Methoden zur Informationsrückgewinnung aus Textdokumenten und die dazu benötigten Sprachverarbeitungsschritte.

2.1 Computerlinguistik

Computerlinguistik ist das Fachgebiet, welches zum Ziel hat Computern zu erlauben nützliche Aufgaben durch die Verarbeitung menschlicher Sprache zu erfüllen. Dabei geht es sowohl um die Interpretation von Sprache, als auch um ihre Erzeugung. Praktische Beispiele sind dabei die maschinelle Übersetzung, automatische Dialogsysteme wie Chatbots, und Systeme zur automatischen Informationsrückgewinnung [DJ08].

2.1.1 Informationsrückgewinnung

Informationsrückgewinnung (eng. „Information Retrieval“) (IR) beschreibt Verfahren zur Suche in einer großen Sammlung nach Material in einer unstrukturierten Form, um einen Informationsbedarf zu stillen. Im Allgemeinen werden dabei Suchanfragen (eng. „queries“) durch einen Nutzer an das System gestellt, welches aus einer großen Sammlung die passenden Informationen zurück gibt. Die zurückgegebenen Informationen werden oft „Dokumente“ genannt, unabhängig davon ob es sich um eine einzelne Datei handelt oder nicht [MRS18].

Im Zusammenhang mit der Auflösung von Abkürzungen in Quelltext ist Informationsrückgewinnung aus zweierlei Gründen relevant: zum einen kann Abkürzungsauflösung ein Mittel sein, die gesuchten Informationen als solche zu erkennen, da ein abgekürzter Bezeichner einen Suchbegriff verschleiern könnte. Zum anderen könnten Auflösungskandidaten einer Abkürzung mittels Methoden der Informationsrückgewinnung bestimmt werden.

2.1.2 Token und Terme

Um den teilweise uneindeutigen Begriff „Wörter“ zu vermeiden, wird in der IR oft von **Termen** gesprochen. Die Definition eines Terms kann in verschiedenen Systemen unterschiedlich sein (z.B.: getrennte oder vereinigte Betrachtung von Wörtern mit Bindestrich). Konkreter unterscheidet wird in der IR zwischen einem *Token*, also einer Instanz einer Zeichensequenz, dem *Typ*, also der Klasse aller Token, welche diese Zeichensequenz enthalten,

und einem *Term*, welcher der Typ im Wörterbuch, oder eine normalisierte Form mehrerer semantisch gleichbedeutender Typen [MRS18] ist, unterschieden.

Um eine konkretere Vorstellung von Termen, beziehungsweise Termini zu haben, hilft die sprachwissenschaftliche Betrachtung: „Benennung (Terminus, Fachausdruck): sprachliche Bezeichnung eines Allgemeinbegriffs aus einem Fachgebiet“ [DIN11]. Diese können Einwortbenennungen und Mehrwortbenennungen sein. In sofern mehrere Worte benötigt werden, um ein Konzept präzise zu benennen, gilt dies also auch als ein Terminus [APS16]. Diese Betrachtungsweise verdeutlicht das Verhältnis von Termen zu ihren, gegebenenfalls nicht normalisierten, Typen und zur zu Grunde liegenden Semantik. Außerdem stellt sich hier die Frage, ob Token gegebenenfalls aus mehreren Wörtern bestehen sollten, wenn sie doch Instanzen von Mehrwortbenennungen sind. Für diese Frage gibt es bislang keine eindeutige Antwort. In dieser Arbeit wird der Begriff „Token“ als eine zusammenhängende Zeichenkette betrachtet, die im Fall eines zusammengesetzten Begriffes nur ein Segment davon beschreibt, welches einen bestimmten Teil der Bedeutung beschreibt.

2.1.3 Spaltung

Da eine Abkürzung sich aus mehreren Wörtern zusammensetzen könnte, und insbesondere Quelltextbezeichner, abgekürzt oder nicht, oft aus mehreren Wörterbuch-Wörtern bestehen, ist es nötig sie in mehrere Bestandteile zu **spalten**. Dabei wird von **harten** und **weichen Wörtern** gesprochen. Harte Wörter sind dabei solche, die eindeutig durch Leer- oder Satzzeichen getrennt sind, während weiche Wörter die Bestandteile solcher Wörter sind, welche eine eigene semantische Bedeutung haben. Diese weichen Wörter sind also nach der Definition aus Abschnitt 2.1.2 auch als Token zu bezeichnen, da sie die Segmente eines zusammengesetzten Begriffes sind, welche einen bestimmten Teil der Bedeutung beschreiben.

2.1.4 Stoppwörter

Sogenannte **Stoppwörter** sind Terme, welche aus Sicht der IR selbst keine Information enthalten, sondern wie Satzzeichen entweder ignoriert, oder als Trennsymbol wahrgenommen werden können. In komplexeren Systemen können sie allerdings auch genutzt werden, um Beziehungen zwischen den anderen Termen herzustellen. Gemeint sind beispielsweise Wörter wie „und“, „das“, „sodass“ u.ä. [MRS18].

2.1.5 Normalisierung

Aus grammatikalischen Gründen werden Wörter in einem Fließtext oft gebeugt. Wird nun bei einer Anfrage zur Informationsrückgewinnung nach einem bestimmten Wort gesucht, sind oftmals auch Resultate interessant, welche das Erkennen verwandter Wörter erfordern. Um diese verwandten Wörter als, für den Zweck der Informationsrückgewinnung, als gleichbedeutend zu markieren, ist es hilfreich, sie in eine einheitliche Form zu überführen. Der Vorgang, mehrere verwandte Wörter in eine einheitliche Form zu bringen, wird **Normalisierung** genannt. Es gibt zwei bedeutende Methoden zu Normalisierung von Wörtern [MRS18].

Die Eine ist das Reduzieren von Wörtern auf ihren Wort-Stamm (eng. **Stammwortreduktion**). Dabei wird ein Wort ausschließlich dadurch normalisiert, dass gewisse Teile, insbesondere Suffixe, abgeschnitten werden. Dies ist zwar einfach zu verwirklichen, kann aber sowohl der Präzision als auch der Ausbeute schaden, wenn Wörter entweder so weit gekürzt werden, dass nicht verwandte Wörter zum gleichen Stamm zusammengefasst werden, oder die Beugung mehr als ein Anfügen von Silben bedeutet, und somit gewisse verwandte Wörter anders normalisiert, und somit als einem anderen Stamm zugehörig betrachtet werden [MRS18].

Lemmatisierung hingegen bildet Wörter auf ihr **Lemma** ab, also eine Basis-Form des Wortes, wie sie zum Beispiel in einem Wörterbuch zu finden wäre. Lemmatisierung zielt also darauf ab alle gebeugten Formen eines Wortes zu erkennen, und sie alle auf ein einheitliches Lemma zurückzuführen. Dies vermeidet im Idealfall zwar die oben erwähnten Probleme der Stammwortreduktion, erfordert aber Kenntnisse über die Grammatik, beziehungsweise der verschiedenen Möglichkeiten, wie ein Wort gebeugt worden sein könnte [MRS18].

2.1.6 Tokenisierung

Tokenisierung meint das Überführen eines Dokuments in eine Liste Token, also jene Terme, welche keine Stoppwörter sind. Je nach System können diese Token auch normalisiert werden, um einen besseren Überblick zu erhalten, zum Beispiel durch *Lemmatisierung* oder *Stammwortreduktion*. Jede Anfrage an das Informationsrückgewinnungssystem sollte die gleiche Tokenisierung durchlaufen, wie die zu durchsuchende Sammlung, damit die Abbildung der Token aus der Anfrage und aus den Dokumenten funktioniert [MRS18].

2.1.7 N-Gramme

Ein N-Gramm-Modell ist ein probabilistisches Modell, welches in einer Wortfolge das nächste Wort anhand der N - 1 vorangegangenen voraussagt. Dazu wird eine N-Gramm-Grammatik erzeugt. Eine N-Gramm-Grammatik ist eine Datenstruktur, welche für alle möglichen N hintereinander auftretenden Terme die Information beinhaltet, wie oft diese Terme in einem gewissen Korpus hintereinander auftreten [DJ08].

2.1.8 Korreferenz

Sprache dient dazu, Konzepte zu beschreiben, die selbst nicht zwingend in der Sprache stattfinden, sondern zum Beispiel in der Realität. Somit bilden Terme auf diese nicht linguistischen Konzepte ab, anders gesagt; sie referenzieren darauf. Bei der IR ist das Ziel des Nutzers oft nicht einen bestimmten genauen Wortlaut zu finden, sondern Informationen über ein, aus Perspektive des Textes, abstraktes, beziehungsweise ein nicht linguistisches Konzept. Wird also ein anderer Wortlaut gefunden, als jener, welchen der Nutzer in seiner Anfrage verwendet hat, kann die Ausbeute seiner Suchanfrage gesteigert werden, indem auch solche Dokumente zurückgegeben werden, in welchen das gleiche Konzept in anderer Formulierung vorkommt. Diese Abbildung zweier verschiedener Terme auf das gleiche Konzept wird Korreferenz genannt [DK00].

Im Zusammenhang mit Quelltext sind diese nicht linguistischen Konzepte, welche in der Sprache durch Quelltextbezeichner beschrieben werden, die Entitäten des Programms. Korreferenzen treten im Quelltext somit zum Beispiel auf, wenn verschiedene Bezeichner auf das gleiche Objekt oder den gleichen Speicherort verweisen, zum Beispiel wenn ein Objekt einer Methode als Parameter übergeben wird, sodass im Methodenrumpf ein anderer Bezeichner auf das gleiche Objekt verweist.

2.2 Quelltext

Es gibt eine Vielzahl von Programmiersprachen und einige verschiedene Familien von Programmiersprachen mit bestimmten Eigenschaften und Funktionsweisen. Die heute wohl verbreitetste Programmiersprachen-Familie ist die der Objekt-orientierten Programmiersprachen. Über deren Syntax lässt sich einiges für die Zwecke dieser Arbeit verallgemeinern.

Grundsätzlich bestehen Objekt-orientierte Programme aus Klassen, Objekten und Methoden. Klassen bieten eine Vorlage zur Erzeugung von Objekten und enthalten verschiedene Methoden. Klassen definieren einen Typ, der wiederum der Typ der Objekte ist, die sie

erzeugen, und der Rückgabe- oder Parameter-Typ einer Methode sein kann. Neben den Methoden kann eine Klasse außerdem eine Menge von Objekten enthalten, die ihre Attribute genannt werden. Klassen können auch voneinander erben. In diesem Fall enthält die erbende Klasse einige der Methoden und Attribute der Eltern-Klasse. Welche geerbt werden und welche nicht, bestimmt ein Zugriffsmodifikator. Außerdem können geerbte Methoden überschrieben werden, und weitere Methoden und Attribute hinzugefügt werden [Weg18].

Zudem verfügt jeder dieser Bausteine eines Objekt-orientierten Programms über einen **Bezeichner**, welcher mindestens innerhalb eines bestimmten Kontext eindeutig sein muss, da der Übersetzer (eng. compiler) diese Bezeichner nutzt, um sie auf tatsächlich durchzuführende Operationen und zu verarbeitende Daten abzubilden. Ebenso nutzten menschliche Leser des Quelltextes diese um die einzelnen Bausteine zu unterscheiden, oder entnehmen dem Bezeichner selbst eine bestimmte Information über die Intention des Entwicklers.

Da die Entitäten, welche durch Quelltextbezeichner beschrieben werden, oftmals nicht in einem einzelnen Token erklärt werden können, werden Quelltextbezeichner oft aus mehreren Token zusammengesetzt. Zwei typische Schreibweisen von zusammengesetzten Bezeichnern sind das Trennen mit Unterstrich (eng. „snake case“) und die Verwendung des Binnenmajuskels (eng. „camel case“) [BDLM09]. Binnenmajuskelschreibweise meint das Schreiben eines zusammengesetzten Begriffes, indem an den Grenzen zwischen den Token der erste Buchstabe eines neuen Token groß geschrieben wird.

2.3 Worteinbettungen

Eine Worteinbettung assoziiert einen Term in einem Vokabular mit einem Vektor. Die Idee dahinter ist, dass die Bedeutung eines Wortes durch seine Beziehung zu anderen Wörtern in einem Vokabular modelliert werden kann [Har54]. Die Eigenschaften eines Terms werden als Dimensionen eines Vektors modelliert, sodass die geometrische Beziehung zwischen den Vektoren die semantische Beziehung zwischen den Termen modelliert. Eine Menge solcher Vektoren, welche auf dem gleichen Korpus trainiert wurden, wird Worteinbettungsmodell genannt. Ein Worteinbettungsmodell wird trainiert, indem ein Vektorraum aufgespannt wird, welcher jedem Term einen Vektor zuweist, sodass zwei Vektoren einander numerisch möglichst ähnlich sind, wenn die durch diese Vektoren repräsentierten Terme im Korpus relativ häufig nahe beieinander auftreten, und umgekehrt. Gleichzeitig soll die Zahl der Dimensionen dieses Vektorraums weit geringer sein als die Größe des Vokabulars. Trainiert werden solche Modelle in aller Regel mittels neuronaler Netze. Eine Ähnlichkeitsfunktion über diese Vektoren soll ausdrücken, wie wahrscheinlich die entsprechenden Wörter in der Nähe zueinander auftreten [BDVJ03]. Ein besonders anschauliches Beispiel für die modellierten Beziehungen, aus einem von Mikolov et al. trainierten Modell [MKB$^+$10], ist die Beziehung zwischen den Vektoren der Wörter „King“, „Queen“, „Man“ und „Woman“. Die Vektorrepräsentation von „Man“ von der von „King“ zu subtrahieren, und die von „Woman“ darauf zu addieren, ergibt einen Vektor der, gemessen an der Cosinusähnlichkeit, dem von „Queen“ sehr nahe ist. Das Modell beschreibt also, wie auch in Abbildung 2.1 illustriert wird, treffend das Geschlecht als die Beziehung, welche den Unterschied zwischen einem König und einer Königin macht [MYZ13].

2.3.1 Cosinusähnlichkeit

Eine typische Funktion für die Ähnlichkeit zweier Vektoren, welche auch in der Nutzung von Worteinbettungen Anwendung findet, ist die Cosinusähnlichkeit. Hierbei wird der Cosinus des Winkels als Maß für die Ähnlichkeit betrachtet, da der Cosinus die willkommene Eigenschaft hat, für identische Vektoren 1, und für orthogonale Vektoren 0 zu betragen

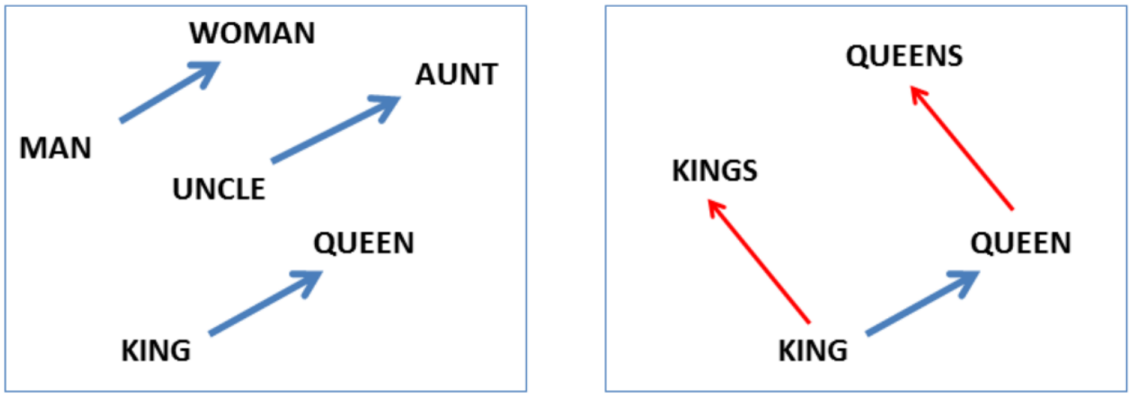

Abbildung 2.1: Beispielgrafik aus [MYZ13]: die blauen Pfeile stellen die Beziehung „Geschlecht“ dar, die roten Pfeile stellen die Beziehung „Plural“ dar.

[Sin01]. Diese Ähnlichkeitsfunktion kann beispielsweise genutzt werden, um zu evaluieren welcher Term, aus einer Auswahl von Kandidaten, in einem bestimmten Kontext am wahrscheinlichsten ist, da er möglichst vielen Token im Kontext ähnelt.

3 Projektumgebung

Die Implementierung der für diese Arbeit entwickelten Software wurde im Rahmen des *INDIRECT*-Projektes [Hey19] entwickelt. *INDIRECT* steht für „Intent-Driven Requirements-to-Code Traceability“ und ist ein Projekt zur Lösung des Problems der Abbildung von Quelltext auf die jeweiligen Anforderungen, welche ein gewisser Teil des Quelltextes erfüllt. *INDIRECT* verwendet dazu einen Absichts-getriebenen Ansatz, indem es die Absichten des Entwicklers aus Anforderungen und Quelltext modelliert, diese Absichten aufeinander abbildet, und so den Zusammenhang zwischen Quelltext und Anforderungen modelliert. Diese Absichtsmodelle werden als Graphen modelliert. *INDIRECT* wurde auf Basis der Architektur *Programming Architecture for Spoken Explanations (PARSE)* [WT15] implementiert. *PARSE* funktioniert auf Basis von Agenten, welche parallel oder sequenziell auf einen zentralen Graphen zugreifen. Indem diese Agenten den Graphen verändern oder mit neuen Informationen anreichern, beeinflussen sie das Verhalten der parallel oder nach ihnen arbeitenden Agenten. Dieser Agenten-basierte Ansatz bietet eine einfache Erweiterung um neue Funktionalitäten durch die Implementierung weiterer Agenten.

Um diese Architektur für die Zwecke von *INDIRECT* einzusetzen, wird, jeweils auf Seite der Anforderungen und des Quelltextes, ein Absichtsmodell in Form eines Graphen angelegt. In diesem werden die Elemente des jeweiligen Artefaktes als Knoten und die se-

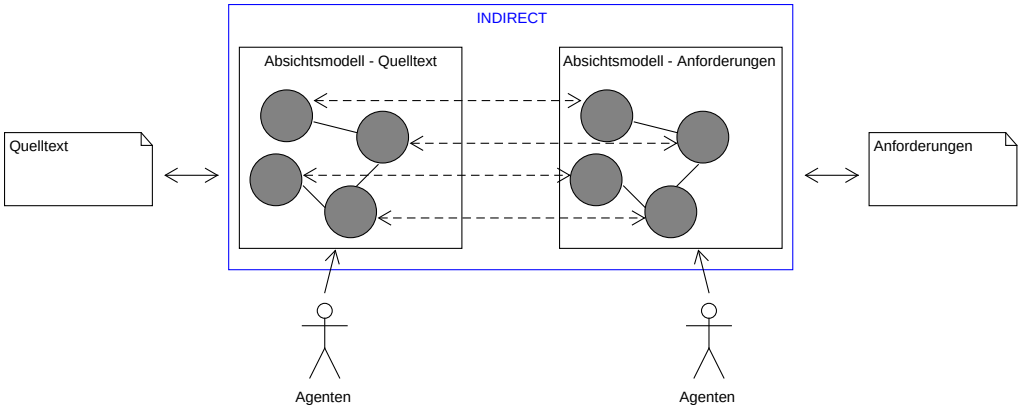

Abbildung 3.1: Skizze der Funktionsweise von *INDIRECT*

mantischen Beziehungen zwischen diesen Elementen als Kanten repräsentiert. Diese beiden Absichtsmodelle können wiederum aufeinander abgebildet werden, um somit die Rückverfolgung zwischen Quelltext und Anforderungen zu gewährleisten. Auf Quelltextseite wird das Absichtsmodell zunächst in Form eines abstrakten Syntaxbaumes angelegt. Dieser wird anschließend durch die Agenten dergestalt verändert und angereichert, dass er die Absicht auf Quelltextseite modelliert. Hierbei gibt es verschiedene Knoten- und Kantentypen, welche die verschiedenen Elemente eines Quelltextes modellieren, indem sie jeweils verschiedene Attribute haben, wie Quelltext, Kommentare, Quelltextzeile, Name des Elements und ähnliches. Abbildung 3.1 bietet eine grobe Illustration der Architektur von *INDIRECT*.

Im Rahmen dieser Arbeit wird nur der Graph auf Quelltextseite genutzt. Dieser wird durch einen Agenten um weitere Knoten- und Kantentypen, sowie neue Attribute für die existierenden Knotentypen, erweitert. Konkret wird ein Knotentyp definiert, welcher einen Quelltextbezeichner, die Quelltextdatei und -Zeile dessen Auftretens, und dessen Auflösung als Attribute beinhaltet. Dazu wird ein Kantentyp definiert, welcher das Quelltext-Element, in welcher der Bezeichner auftritt, mit dem Bezeichner verbindet. Außerdem werden die Knotentypen für Anweisungen, Quelltext-Blöcke, Methoden und Klassen jeweils um ein Attribut erweitert, welches einen Kontext für die Suche nach Abkürzungsauflösungskandidaten darstellt. Somit wird einem anderen Agenten die Durchführung der Abkürzungsauflösung zu ermöglichen. Im Rahmen dieser Arbeit wurden verschiedene Abkürzungsauflösungsverfahren jeweils als ein Agent implementiert. Die Ergebnisse der Auflösung werden in den Graphen zurückgeschrieben.

4 Verwandte Arbeiten

Dieses Kapitel beschreibt einige vorangegangene Forschungsarbeiten, welche dazu dienen den Stand der Forschung zum Zeitpunkt dieser Arbeit wiederzugeben, und dazu die Entwurfsentscheidungen, welche hier getroffen wurden, zu begründen, zu inspirieren und abzugrenzen.

4.1 Automatisches Auflösen von Abkürzungen in Quelltext

Lawrie et al. befassen sich in „Extracting Meaning from Abbreviated Identifiers“ [LFB07] erstmals mit der Transformation von Quelltext durch die Auflösung von Abkürzungen. Ihr Ziel ist es, das Verständnis beim Lesen von fremdem Quelltext und die Effektivität von IR-basierten Systemen, welche, mit Hilfe natürlicher Sprache, Informationen aus dem Quelltext gewinnen wollen, zu verbessern. Einleitend wird auf das Problem der Auflösung von Abkürzungen in Quelltext eingegangen, und zwar in drei Schritten: (1) Erkennung der Abkürzung, (2) finden von Auflösungskandidaten und (3) Mehrdeutigkeitsauflösung zur Wahl des richtigen Kandidaten. Außerdem wird ein mögliches Vorgehen für den Gesamtprozess beschrieben. Vorgeschlagen wird das Identifizieren üblicher und unüblicher Wörter mittels Wahrscheinlichkeitsverteilungen wie relativer Entropie, Auflösung von Abkürzungen mittels Stellvertretersymbol-Auflösung, und Auswahl des besten Kandidaten mittels eines lernenden Algorithmus. Im Spezialfall von Multiwort-Bezeichnern soll außerdem ein Verfahren zur Mehrdeutigkeitsauflösung, welches aus der Maschinenübersetzung entlehnt wird, zum Einsatz kommen. Vorgestellt wird nach der Einleitung ein Ansatz für den zweiten Schritt der Auflösung. Die Auflösung wird in zwei Schritten ausgeführt: Spaltung und Auflösung. Für das Spalten von Multiwort-Bezeichnern in weiche Wörter wird ein gieriger Algorithmus eingesetzt, welcher rekursiv die längsten Präfixe und Suffixe sucht, welche Wörterbuch-Wörter darstellen. Für die Auflösung werden vier Listen von Kandidaten verwendet: Erstens natürlichsprachliche Wörter aus dem Quelltext, was sowohl Wörter aus den Kommentaren vor und innerhalb der Methode, als auch solche in Bezeichnern innerhalb der Methode, in welcher sich der betrachtete Bezeichner befindet, einschließt. Zweitens Multiwort-Begriffe aus dem gleichen Kontext, welche mit Hilfe eines Begriffsfinders von Feng und Croft [FBC01] gewonnen werden. Aus diesen werden auch Akronyme konstruiert, welche bei der Auflösung von Abkürzungen auf Übereinstimmung überprüft werden. Hinzu kommt ein natürlichsprachliches Wörterbuch. Zuletzt werden Programmiersprachen-spezifische Wörter als Stoppwort-Liste gehalten, sodass diese nicht als Auflösungskandidaten in Frage kommen. Die Auflösung verläuft in zwei Schritten. Im

ersten werden die ersten beiden Wissensquellen, also die Wörter und Multiwort-Begriffe aus dem Quelltext nach Kandidaten durchsucht. Wenn bei diesem Schritt mindestens ein Kandidat gefunden wird, wird Schritt zwei gar nicht erst ausgeführt. Ansonsten werden in Schritt zwei Wörterbuch und Stoppwort-Liste als Wissensquelle genutzt. Ein Wort aus der Wissensquelle gilt als Auflösungskandidat für eine bestimmte Abkürzung, wenn es den gleichen Anfangsbuchstaben wie die Abkürzung hat und alle Buchstaben in der Abkürzung in entsprechender Reihenfolge im Kandidaten vorkommen. Um die Auflösung zu verbessern, wurden die IR-Methoden Stammwortreduktion und Stoppwortentfernung eingesetzt. Wenn mehr als ein Kandidat gefunden wird, wird die Abkürzung nicht aufgelöst. Bei der Evaluation der Methode wurden dreizehn Programme als Probanden herangezogen. Daraus wurden 64 Bezeichner, welche tatsächlich von der Auflösung betroffen waren, zufällig ausgewählt und das Ergebnis manuell inspiziert. Dabei wurde festgestellt, dass bestimmte Abkürzungen besonders schwer aufzulösen sind. Ist die Abkürzung zum Beispiel sehr geläufig, ist es unwahrscheinlich, dass ein Entwickler ihre Auflösung in einen Kommentar in der Nähe beschreibt. Das Spalten von Bezeichnern wird erschwert, wenn ein Bezeichner zufällig eine Zeichenkette enthält, die sich auch als ein anderes Wort interpretieren ließe („argcasematch“ enthält „argc“). Zudem sind gerade einzelne Buchstaben nicht immer als Abkürzungen gemeint, wie x,y und z als Koordinaten oder m und n als Zeilen und Spalten einer Matrix. Auch wurde beobachtet, dass die Korrektheit, sowohl für die Auflösung von Abkürzungen, als auch für die Spaltung von Bezeichnern, von der jeweiligen Länge abhängt. Die Auflösung war für Abkürzungen die ein oder zwei Zeichen lang sind in 57% der Fälle korrekt, in 64% der Fälle bei längeren. Ebenso war die Spaltung für Bezeichner, die ein oder zwei Zeichen lang sind, in 57% der Fälle korrekt, bei längeren in 91% der Fälle. Zuletzt wurde beobachtet, dass Auflösungen für allgemein bekannte Akronyme eher in externer Dokumentation als im Quelltext zu finden sind, und dass das Großschreiben jedes Zeichens für die Erkennung von Akronymen ein hilfreiches Indiz sein kann. Bei der Evaluation des Auflösungsalgorithmus wurden verschiedene Statistiken betrachtet: Es wurde betrachtet, wie gut gewisse Bezeichner erweitert werden, und wie hilfreich dabei die verschiedenen Wissensquellen sind. Es war zum Beispiel auffällig, dass Java-Programme in der Regel weniger Abkürzungen enthielten als C oder C++ Programme, was von Konventionen herrühren könnte, zu deren Einhaltung Java-Entwickler ermutigt werden. Insgesamt hat der Algorithmus etwa 16% aller Bezeichner in den Programmen erweitert, wobei der Anteil je Programm zwischen 3% und 31% schwankte. Eine knappe Hälfte der erweiterten Bezeichner hatte mehrere Auflösungskandidaten. Der Quelltext und die Dokumentation enthalten etwa für 60% der weichen Wörter passende Auflösungen, wobei Quelltext und Kommentare etwa einen gleichen Anteil dieser richtigen Auflösungskandidaten enthalten.

Hill et al. haben mit „AMAP: Automatically Mining Abbreviation Expansions in Programs to Enhance Software Maintenance Tools“ [HFB+08] ein Verfahren entwickelt, welches für die Auflösung von Abkürzungen in Quelltext zunächst in unmittelbarer Nähe zum Bezeichner nach Auflösungskandidaten, dann im Rumpf von Methoden, dann in den Kommentaren der Methode, dann in der ganzen Klasse sucht. Erst wenn dort keine Kandidaten gefunden werden, wird stattdessen im gesamten Programm, dann in der gesamten Java-API nach Auflösungskandidaten gesucht. Es werden verschiedene Abkürzungstypen definiert: Präfixe, Wörter aus denen Buchstaben weggelassen wurden, einzelne Buchstaben, Akronyme, und Kombinationen aus den zuvor genannten. Für die verschiedenen Abkürzungstypen wurden verschiedene Algorithmen entwickelt, welche alle auf regulären Ausdrücken basieren. Jede potentielle Abkürzung wird zuerst als Akronym, dann als Präfix, dann als Wort mit fehlenden Buchstaben, und zuletzt als kombinierter Multiwort-Bezeichner behandelt, um nach Auflösungskandidaten zu suchen. Enthält eine Abkürzung mehrere Vokale hintereinander, wird davon ausgegangen, dass es sich um eine Multiwort-Abkürzung, also ein Akronym oder ein kombiniertes Multiwort, handelt. In jedem dieser Schritte wird zunächst ein Suchmuster für die Abkürzung in Form eines regulären Ausdrucks konstruiert.

Für jeden Abkürzungstyp funktioniert diese Konstruktion anders. Für die Behandlung als Präfix-Abkürzungen besteht der reguläre Ausdruck aus dem Bezeichner, gefolgt von „[a-z]+“, also einer positiven Anzahl weiterer Buchstaben. Der Buchstabe „x“ ist ein Sonderfall: beginnt der Bezeichner mit „x“, wird „e?“, also die Optionen eines „e“s vorne angehängt. Wird der Bezeichner für den Fall verarbeitet, dass Buchstaben in ihm weggelassen wurden, wird das Muster „[a-z]*“, also keine oder eine positive Anzahl weiterer Buchstaben, nach jedem Zeichen des Bezeichners eingefügt. Für Akronyme wird nach jedem Buchstaben des Bezeichners das Muster „[a-z]+[ ]+“, also eine positive Anzahl Buchstaben und mindestens ein Leerzeichen, eingefügt. Auch hier gilt die gleiche gesonderte Behandlung des Buchstaben „x“, hier allerdings an beliebiger Position des Bezeichners. Bei kombinierten Multiwort-Bezeichnern wird das Muster „[a-z]*?[ ]*?“, also beliebiges Auftreten oder Nichtauftreten von Buchstaben und Leerzeichen, nach jedem Buchstaben eingefügt. Dies führt in dieser Form auch dazu, dass nur Buchstaben aus dem ursprünglichen Bezeichner Anfangsbuchstaben der Auflösung sein können. Zudem gibt es jeweils für Einwort- und Multiwort-Abkürzungen unterschiedliche Auflösungsstrategien, die jeweils mit Fallunterscheidungen nach Abkürzungstyp und Länge des Bezeichners weiter aufgeteilt sind. Wird der Bezeichner als Einwort-Abkürzung behandelt, wird in dieser Reihenfolge in den JavaDoc-Kommentaren, den Typ- und Variablen-Namen innerhalb des Methodenrumpfes, dem Methodennamen und den Anweisungen nach Auflösungskandidaten gesucht. Nur wenn der Bezeichner länger als zwei Zeichen ist, wird auch im restlichen Text und in den Kommentaren, in und um die Methode, gesucht. Nur wenn es sich um eine Präfix-Abkürzung handeln soll, welche länger als ein Zeichen ist, wird zuletzt in den Klassen-Kommentaren gesucht. Bei der Suche nach Auflösungskandidaten für Multiwort-Abkürzungen wird in dieser Reihenfolge in den JavaDoc-Kommentaren, den Typ- und Variablen-Namen innerhalb der Methode, in allen Bezeichnern innerhalb der Methode, in den String-Literalen innerhalb der Methode und in den Methoden-Kommentaren nach Auflösungskandidaten gesucht. Nur wenn es sich bei der Abkürzung um ein Akronym handeln soll, werden auch die Klassen-Kommentare durchsucht. Bei Uneindeutigkeiten wird der am häufigsten im Kontext auftretende Kandidat genutzt, wobei gegebenenfalls Kandidaten mit gleichem Wortstamm gesammelt betrachtet werden, oder der beobachtete Kontext erweitert wird, bis die Häufigkeit des Kandidaten im ganzen Programm betrachtet wird. Für die Evaluation wurden 250 nicht natürliche Wörter automatisch aus dem Quelltext extrahiert und manuell zu einer Musterlösung aufgelöst. Diese beinhaltete auch nicht natürliche Wörter, welche keine Abkürzungen sind, und dementsprechend korrekterweise nicht aufgelöst werden sollten. Auf diesen Abkürzungen erzielt das Verfahren eine Genauigkeit von 58,8%.

Der Name *Normalize* des Verfahrens von Lawrie und Binkley rührt von der Idee her, dass ein Vokabular für eine klare Wiedererkennung der Semantik *normalisiert* werden sollte. Das heißt hier, dass die Bezeichner im Quelltext der natürlichen Sprache in der Dokumentation anzupassen sind, damit ein Leser die Funktion des Programms versteht. Die Auflösung von Abkürzungen dient also dazu, die Bezeichnungen der in der Dokumentation erörterten Konzepte und die Bezeichner des sie implementierenden Quelltextes in Übereinstimmung zu bringen. Sie teilen diese Aufgabe in „Expanding Identifiers to Normalize Source Code Vocabulary“ [LB11] in zwei Schritte. Zuerst werden die Bezeichner gespalten und anschließend die daraus resultierenden Bestandteile der Bezeichner zu natürlichen Wörtern aufgelöst. Zur Spaltung wird ein Verfahren namens *GenTest* [LBM10] verwendet, welches zugleich die eigenen Spaltungen nach Plausibilität bewertet. *GenTest*, beschrieben in „Normalizing Source Code Vocabulary“ [LBM10] von Lawrie et al., spaltet die Bezeichner zunächst anhand von Binnenmajuskelschreibweise oder Trennung mit Unterstrich in harte Wörter. Dann spaltet es diese harten Wörter in alle möglichen weichen Wörter, und bewertet anschließend die so entstandene Liste. Die Bewertung erfolgt anhand verschiedener Charakteristika der weichen Wörter, wie deren Anzahl im harten Wort, de-

ren durchschnittliche Länge, oder die Länge des längsten weichen Wortes im harten Wort. Hinzu kommen externe und interne Informationen, was statistische Informationen über die weichen Wörter im Wörterbuch, beziehungsweise im Quelltext meint. Ein Betrachten aller möglichen Spaltungen würde Kosten verursachen, die exponentiell mit der Wortlänge steigen. Deshalb werden nur die besten zehn Kandidaten, beziehungsweise später nur der beste Kandidat berücksichtigt. Für jedes der weichen Wörter in einer Spaltung werden dann mögliche Auflösungen, mittels Stellvertretersymbol-Auflösung, aus einem Lexikon ermittelt. Die plausibelste Auflösung wird mittels einer Bewertung ermittelt, welche aus den Bewertungen der Gleichheit aller Teilbezeichner mit ihren Auflösungskandidaten addiert wird, multipliziert mit einer Wertung der Wahrscheinlichkeit des gemeinsamen Auftretens aller Teilauflösungen. Als Lexikon werden hier der Quelltext und dessen Kommentare selbst, die Dokumentation, das Nutzerhandbuch und beliebige Kombinationen dieser drei in verschiedenen Experimenten verwendet. Die Evaluation des Auflösungsverfahrens kam zu dem Schluss, dass das Verfahren am besten funktioniert, wenn nur Auflösungskandidaten basierend auf dem am besten bewerteten Spaltungskandidaten von *GenTest* berücksichtigt werden, nicht basierend auf den besten zehn Spaltungskandidaten von *GenTest*. Das Lexikon, mit welchem die besten Auflösungsergebnisse erzielt wurden, war die Dokumentation. *Normalize* löst 53% aller Bezeichner im Probanden-Programm *which* vollständig richtig auf, in *a2ps* nur 32%. Eine Frage, welche die Autoren hier stellen, aber deren Antwort sie weitgehend für die Zukunft offen lassen, ist, ob eine engere Integration von Spaltung und Auflösung für den einen oder anderen Teilschritt von Vorteil sein könnte, also ob zum Beispiel die Spaltung bessere Ergebnisse liefern könnte, wenn sie mögliche Auflösungen für die Auswahl in Erwägung ziehen würde.

Corazza et al. sehen Verbesserungspotential, gegenüber den ihnen vorangehenden Lösungsansätzen zum Thema Abkürzungsauflösung, in der Auflösung von Mehrdeutigkeiten in der Auflösungsauswahl, durch die Nutzung mehrerer Kontexte, und deren Priorisierung anhand ihrer Nähe zur konkretesten Domäne. Um dies zu bewerkstelligen, nutzen sie in ihrer Arbeit „LINSEN: An efficient approach to split identifiers and expand abbreviations“ [CDMM12] den Baeza-Yates- und Perleberg-Algorithmus [BYP96] für approximative String-Anpassung. Dessen lineare Laufzeit hält die Mehrkosten der Verwendung mehrerer Lexika in Grenzen. Die verwendeten Kontexte sind der Quelltext und dessen Kommentare, (1) beschränkt auf die Datei, in der sich der betrachtete Bezeichner befindet, (2) aus dem gesamten Projekt, (3) ein Wörterbuch von IT- und Programmier-Begriffen und (4) ein englisches Wörterbuch. Außerdem wird jede Abkürzung mit einer Liste allgemein bekannter Abkürzungen abgeglichen. Die Auflösung eines Bezeichner erfolgt in zwei Schritten: (1) der Bezeichner wird nach natürlichsprachlichen Token durchsucht. Gefundene Token werden auf die entsprechenden natürlichen Wörter abgebildet und sind damit fertig behandelt. Der Bezeichner wird somit, falls natürlichsprachliche Token und nicht natürlichsprachliche Token in ihm vorkommen, in diesem Schritt auch an den natürlichsprachlichen Token gespalten. (2) Der Auflösungs- und Abbildungs-Algorithmus geht davon aus, dass alle nicht im ersten Schritt abgebildeten Token Abkürzungen sind. Erkennt er ein Token als Multiwort-Bezeichner, erfolgt hier eine weitere Spaltung. Diese basiert auf der Annahme, dass die einzelnen Token, welche den Multiwort-Bezeichner zusammensetzen, anderswo in der selben Datei auftauchen. Für beide Schritte wird für jeden Bezeichner ein Matching-Graph aufgebaut. Der Graph enthält einen Knoten für jedes Zeichen im Wort, und eine Kante für jedes ungefähre Übereinstimmen mit einem Wörterbuch-Wort. Die Kanten verbinden also die Knoten, welche Buchstaben des Bezeichners darstellen, und sind mit möglichen Auflösungen der Token gekennzeichnet, welche mit dem Buchstaben anfangen, welcher im Ursprungsknoten der Kante liegt. In der Regel verbinden die Kanten also die Buchstaben am Anfang der Token. Um sicher zu stellen, dass es immer einen Ergebnispfad gibt, gibt es auch Kanten zwischen den Buchstaben eines Tokens im Bezeichner, diese sind allerdings mit sehr hohen Gewichten versehen, da diese meistens vernachlässigt

werden sollen. Die Auflösungskandidaten, welche die Kanten darstellen, werden mit Hilfe des BYP gefunden. Der BYP führt einen approximativen Wort-Vergleich durch, indem er den Unterschied zwischen zwei Wörtern anhand von sogenannten Fehlern misst, nämlich Löschungen, Einfügungen oder Substitutionen von Buchstaben, die nötig wären, damit die Wörter identisch sind. Der BYP-Algorithmus nutzt eine Toleranzfunktion für die Entscheidung, ob die Zahl der Fehler zwischen zwei Wörtern es zulässt, sie als ähnlich, und damit hier als Kandidaten, zu betrachten. Die Toleranzfunktion bestimmt die maximale Zahl an Fehlern, welche in der Abbildung auf ein natürliches Wort vorliegen dürfen. Im ersten Schritt, dem Herausfiltern natürlicher Token aus dem Bezeichner und der daraus resultierenden ersten Spaltung, ist die Toleranz Null, da natürliche Wörter erkannt werden sollen. Nachdem so die Auflösungskandidaten für ein Token gefunden sind, wird der als richtig erachtete Kandidat mittels des Matching-Graphen ermittelt. Kantengewichte im Graphen sind im ersten Schritt andere als im zweiten, dem Auflösungsschritt, und hängen von der Gesamtzahl der Vorkommnisse jedes natürlichen Wortes in der Wissensquelle ab. Diese Kantengewichte sind so gewählt, dass sie längere Wörter und Wörter aus Wissensquellen der Domäne bevorzugen. Diese Gewichtung eines jeden Wortes wird während des Aufbaus des Wörterbuchs aus den Wissensquellen bestimmt und kann in konstanter Zeit gelesen werden. Als Ergebnis werden die Label der Kanten, also die natürlichen Wörter, des kürzesten Pfades, welcher mittels des Dijkstra-Algorithmus ermittelt wird, zurückgegeben. Dieses Auflösungsverfahren verwendet die vier zuvor aufgelisteten Wissensquellen, priorisiert entsprechend der Reihenfolge in der sie aufgelistet sind. Im Auflösungsschritt wird zunächst geprüft, ob ein Token in der Liste wohl bekannter Abkürzungen auftritt. Wenn nicht, wird das Token auf ungefähre Übereinstimmung mit den natürlichen Wörtern aus verschiedenen Wissensquellen überprüft. Es werden nur Auflösungen in Betracht gezogen, welche mittels bestimmter Abkürzungsstrategien zum betrachteten Token verkürzt werden können. Es werden zum Beispiel nur Löschungen in Mitte oder Ende des Wortes in Erwägung gezogen, wobei Löschungen am Ende für wahrscheinlicher erachtet werden, und Löschungen von Vokalen für wahrscheinlicher als Löschungen von Konsonanten. Auch wird davon ausgegangen, dass Abkürzungen mit mehr Vokalen als Konsonanten Präfixe sind. Diese Annahmen sind in verschiedene Toleranzfunktionen, die in verschiedenen Momenten des Auflösungsschrittes eingesetzt werden, eingebettet. Evaluiert wurde das automatische Verfahren auf zwei Probanden-Programmen, *JHotDraw* und *Lynx*, aus denen jeweils Musterlösungen von 957, beziehungsweise 3.085 Bezeichnern durch Madani et al. [MGDP+10] manuell aufgelöst wurden. Diese Musterlösung wurde auch genutzt, um das Verfahren mit dem von Madani et al. zu vergleichen. Außerdem wurde das Verfahren auf den Datensätzen von Lawrie et al. mit deren Spalt-Verfahren namens *GenTest* [LBM10], und deren Abkürzungsauflösungsverfahren namens *Normalize* [LB11] verglichen. Dieser Datensatz bestand aus 487 Bezeichnern aus *which* und 211 Bezeichnern aus *a2ps* [LFB07]. Bei der Spaltung, welche anhand der durch Madani et al. erzeugten Musterlösungen evaluiert wurde, wurde eine Genauigkeit von zwischen 80,3% auf Lynx und 94,9% auf JHotDraw erzielt. Evaluiert anhand der Musterlösung von Lawrie et al., betrug die Genauigkeit der Spaltung zwischen 50,3% auf *a2ps* und 64,6% auf *which*, bezogen auf ganze Bezeichner, beziehungsweise zwischen 75,1% auf *a2ps* und 78,3% *which*, bezogen auf die einzelnen weichen Wörter. Bei der Auflösung von Abkürzungen in Bezeichnern wurde, bezogen auf die Musterlösung von Madani et al., eine Genauigkeit von 94,5% auf Lynx und von 99,1% auf JHotDraw erzielt. Die Genauigkeit der Auflösung betrug zwischen 56,6% auf *a2ps* und 56,7% auf *which*, bezogen auf ganze Bezeichner, beziehungsweise zwischen 71,3% auf *a2ps* und 83,5% auf *which*, bezogen auf die einzelnen weichen Wörter, beides anhand der Musterlösung von Lawrie et al. evaluiert. Auch wurde evaluiert, wie gut das Verfahren im Bezug auf gewisse Abkürzungstypen abschneidet. Für Akronyme wurde es mit einer Genauigkeit von 36% für besonders schlecht befunden. Die Autoren liefern als mögliche Erklärung, dass, gerade für diese Abkürzungstypen, die Priorisierung des engmaschigen Kontextes unangebracht

sein könnte.

Guerrouj et al. stellen in „TRIS: a Fast and Accurate Identifiers Splitting and Expansion Algorithm“ [GGG+12] ein Verfahren vor, welches Bezeichner in einem Baum anordnet, und die Spaltung und Auflösung von abgekürzten Bezeichnern als Minimierungsproblem, genauer als Suche nach dem kürzesten Pfad in einem Graphen, darstellt. Die Autoren beschreiben den Vorgang, wie abgekürzte Quelltextbezeichner entstehen, so, dass Entwickler gewisse Transformationsregeln auf natürliche Wörter anwenden. Auf dieser Perspektive basiert ihre Idee, diese Transformation entlang der Pfade eines Baumes zurück zu verfolgen, dessen Blätter die Buchstaben der Abkürzung sind. Sie bezeichnen das Problem als optimale Spaltungs-Auflösung (OSE). Die Eingabe für das Problem bilden (1) der zu spaltende/aufzulösende Bezeichner, (2) ein Wörterbuch und (3) der Quelltext, aus welchem der Bezeichner stammt. Als Eingabe für die Baum-Erzeugung dienen eine Menge von Wörtern, genauer eine Ontologie aus der Domäne des Programms, und der Quelltext, dessen abgekürzte Bezeichner gespalten und erweitert werden sollen. Für jedes Wort und jeden Transformationstyp wird eine Menge aller Transformationen erzeugt, die mit diesem Transformationstyp aus dem Wort entstehen können. Diese Transformationen werden zu einem gerichteten Baum angeordnet, dessen Knoten Buchstaben repräsentieren. Es wird zwischen vier Arten von Transformationen unterschieden: erstens (1) „Null“ - das Wort bleibt wie es ist, zweitens (2) das Entfernen von Vokalen, drittens (3) Entfernen eines Suffixes, insbesondere solche wie „ing“ oder „tion“, es könnte hier also auch von Stammwortreduktion gesprochen werden. Die vierte und letzte Transformation ist (4) das Behalten der ersten n Buchstaben, also eine Präfix-Abkürzung. Ein Lösungskandidat, ergo ein Element des Suchraums, entspricht einer Spaltungs-Auflösung, und wird aus einer Reihe von Transformationen zusammengesetzt. Die Kosten eines Kandidaten sind die Summe der Kosten der Transformationen, die er benötigt. Die Kostenfunktion einer Transformation wiederum setzt sich zusammen aus der Frequenz des Zielwortes, indem häufiges Auftreten die Kosten senkt, und einer Konstante je Transformationstyp, davon ausgehend, dass einige Abkürzungsstrategien beliebter bei Entwicklern und damit wahrscheinlicher sind. Diese Kosten stellen also die Gewichte der Kanten im oben beschriebenen Graphen dar. Evaluiert wurde nur die Spalt-Fähigkeit des Verfahren. Zur Evaluation wurden insgesamt knapp 5.000 Bezeichner aus vier Programmen manuell zu einer Musterlösung erweitert und außerdem die Musterlösung aus der Arbeit von Lawrie et al. [LB11] verwendet. Außerdem werden Wörterbücher aus Begriffen der Quelltexte von zwei der betrachteten Programme angelegt, mit jeweils etwa 2.500 Begriffen. Die Analyse berücksichtigt keine Quelltext-Abkürzungen, die zu einem einzelnen Wort aufzulösen sind, da hier keine Spaltungs-Auflösung benötigt wird. Außerdem werden solche Abkürzungen heraus gefiltert, die Präfixe aus nur einem oder zwei Buchstaben enthalten, da solche auf praktisch jedes Wort abgebildet werden könnten, welche mit diesem Buchstaben beginnen. Gemessen wurde die Korrektheit der Spaltung von Bezeichnern, gemessen in Präzision, Ausbeute und F-Maß. Dies wurde auf verschiedenen Datensätzen mehrfach evaluiert. Darunter auch der oben beschriebene Gesamtdatensatz. Bei letzterem erreichte *TRIS* eine Genauigkeit von 86%.

In „Bayesian Unigram-Based Inference for Expanding Abbreviations in Source Code“ [AXX17] stellen Alatawi et al. ein Verfahren zum Auflösen von Abkürzungen in Quelltext vor, indem sie die Unigramm-statischen Eigenschaften der Abkürzung als Auswahlkriterium für den wahrscheinlich richtigen unter mehreren Auflösungskandidaten heranziehen. Der Begriff Unigramm bezieht sich hierbei auf den N-Gramm-Begriff, welcher einen Begriff beschreibt, der aus N aufeinander folgenden Wörtern besteht. Mit einem Unigramm ist also ein einzelnes Wort gemeint. Ein abgekürzter Bezeichner, welcher einen Multiwort-Begriff meint, kann als Zusammensetzung aus N Unigrammen betrachtet werden. Ziel ist herauszufinden, wo der Bezeichner in seine weichen Wörter zu spalten ist, und auf welche Unigramme diese jeweils am wahrscheinlichsten abzubilden sind. Hier wird anstelle von

weichen Wörtern von *Abrreviation N-gram Unique Part*, kurz *ANUP* gesprochen, was als Synonym zu den schon etablierten weichen Wörtern betrachtet werden kann. Sowohl für die Spaltung als auch für die Auflösung werden die betrachteten Unigramm-Grammatiken verwendet. Die richtige Spaltung eines Bezeichners wird zusammen mit dem plausibelsten Auflösungskandidaten entschieden. Die Auflösungskandidaten stammen aus dem Quelltext. Da auch natürlichsprachliche Wörter im Quelltext in zusammengesetzten Bezeichnern auftreten können, wird ein Werkzeug namens *WordSegment 0.6.2* von Grant Jenks [Jen14] zur Spaltung solcher Bezeichner verwendet, um die einzelnen Kandidaten aus ihnen zu gewinnen. Gegeben eine Spaltung des abgekürzten Bezeichners, wird die Qualität eines Paares, aus einem Auflösungskandidaten und einem weichen Wort, anhand von drei Eigenschaften der Paare von Abkürzungen und Kandidaten, basierend auf den statistischen Eigenschaften der Kandidaten-Unigramme, evaluiert: Erstens (1) der „Abkürzungs-Strategie-Typ“, welcher beschreibt wie der Entwickler das weiche Wort aus dem ursprünglichen Begriff abgekürzt hat. Da jedes weiche Wort einzeln betrachtet wird, und die Autoren davon ausgehen, dass jede Abkürzung den Anfangsbuchstaben des Ursprungswortes enthält, sind die einzigen beiden für sie in Frage kommenden Typen Präfixe, hier konsekutive Abkürzungen genannt, und Wörter in denen Buchstaben, außer der Anfangsbuchstabe, weggelassen wurden, hier nicht konsekutive Abkürzungen genannt. Zweitens (2) der eingesparte Aufwand beim Tippen. Die Autoren gehen davon aus, dass Entwickler Wörter mit höherer Wahrscheinlichkeit abkürzen, wenn diese einen hohen Tippaufwand darstellen, und somit Auflösungskandidaten, welche einen höheren Tippaufwand darstellen, wahrscheinlicher sind. Und drittens und zuletzt (3) der Abstand des Auflösungskandidaten von der Abkürzung im Quelltext. Die Autoren gehen davon aus, dass Entwickler die Wörter, welche sie in Bezeichnern abkürzen, in der Nähe des Bezeichners ausschreiben, Kommentare eingeschlossen. Somit werden näher gelegene Kandidaten als wahrscheinlicher erachtet. Außerdem wird die Wahrscheinlichkeit eines Kandidaten an sich, basierend auf seiner Frequenz im Quelltext und in einer externen Wissensquelle, in seine Wahrscheinlichkeit als Auflösungskandidat mit einbezogen. Als Wissensquelle für diese Frequenz werden entweder ein natürlichsprachlicher Datensatz des „Linguistics Data Consortium“ [BF06], welcher 2006 von Google zur Verfügung gestellt wurde, basierend auf einer Billion Token aus frei zugänglichen Websites, oder ein Software-basierter Datensatz, welcher aus über einer Milliarde Unigrammen aus 700.000 Open-Source-Projekten generiert wurde [XXA$^+$18] herangezogen. Um den gesamten Bezeichner aufzulösen, muss der wahrscheinlichste zusammengesetzte Kandidat für den gesamten Bezeichner bestimmt werden. Hierzu werden die Wahrscheinlichkeiten für alle einzelnen ANUP-Kandidat-Paare kombiniert. Als die richtige Spaltung des Bezeichners in weiche Wörter wird die angesehen, in welcher die kombinierte Wahrscheinlichkeit der jeweils besten Kandidaten am höchsten ist. Spaltung und Auflösung des gesamten Bezeichners werden also gleichzeitig entschieden. Als Probanden für die Evaluation wurden 531 Abkürzungen zufällig aus acht Open-Source-Programmen extrahiert und manuell aufgelöst. Neben diesem Vergleich der allgemeinen Bestleistung wurde auch der Einfluss unter den beiden Wissensquellen für die Unigramm-Grammatik beobachtet. Die Genauigkeit bei Einsatz des Software-basierten Datensatzes beträgt etwa 83,62%, bei Einsatz des natürlichsprachlichen Datensatzes nur 80,23%. Die Ergebnisse beim Auflösen von Akronymen oder Abkürzungen, welche aus einem einzelnen Buchstaben bestehen, sind verhältnismäßig schlecht.

Alatawi et al. setzen diese Idee in „The Expansion of Source Code Abbreviations Using a Language Model“ [AXY18] fort. Sie verwenden ein Bigramm-basiertes Inferenz-Modell, welches die gleichen Unigramm-statistischen Eigenschaften nutzt wie oben. Im Wesentlichen besteht der Ansatz aus drei Schritten: (1) Segmentieren des Bezeichners, (2) Generieren von Begriffen als Auflösungskandidaten für jede mögliche Sequenz an Segmenten und (3) Auswählen des besten Kandidaten. Als Segmentierungs-Kandidaten werden zunächst alle Möglichkeiten, wie der Bezeichner segmentiert werden könnte, in Erwägung

gezogen. Für jede Segment-Sequenz wird eine Menge an Begriffs-Kandidaten erzeugt, indem für jedes Segment Unigramm-Kandidaten gefunden und daraus alle möglichen Begriffe kombiniert werden. Die Auswahl des besten Kandidaten basiert auf den gleichen drei Unigramm-basierten Strategien wie in der zuvor beschriebenen Arbeit von Alatawi et al. [AXX17] : Erstens (1) auf dem „Abkürzungs-Strategie-Typ“, zweitens (2) auf dem eingesparten Aufwand beim Tippen, und drittens (3) auf dem Abstand zwischen der Abkürzung und dem Auflösungskandidaten im Quelltext. Zusätzlich wird eine Bigramm-Grammatik genutzt, welches die Wahrscheinlichkeit des betrachteten Wortes im Kontext des vorangehenden Wortes betrachtet. Außerdem wird die Frequenz einer Abkürzung innerhalb der betrachteten Quelle berücksichtigt, und zwar dahingehend, dass davon ausgegangen wird, dass die gleiche Abkürzung überall das Gleiche meint, und somit die in diesem Einzelfall beobachtete Wahrscheinlichkeit der Kandidaten nur einen Bruchteil aller Beobachtungen dieser Abkürzung darstellt. Die empirische Auswertung des Verfahrens vergleicht die gleichen Wissensquellen gegeneinander, wie die oben erläuterte Arbeit von Alatawi et al. [AXX17]: Den natürlichsprachlichen Datensatz des „Linguistic Data Consortium“ [BF06] und den Software-basierten Datensatz, welcher aus Open-Soruce Projekten generiert wurde [XXA+18]. Bei der Nutzung von Letzterem wurde das Verfahren mit einer Genauigkeit von 78% für erfolgreicher befunden als unter Verwendung des natürlichsprachlichen mit 69%. Für die Evaluation wurden 100 Abkürzungen aufgelöst.

Jiang et al. wurden vom Erfolg des *LINSEN* Ansatzes [CDMM12] inspiriert noch konkreter auf einzelne Kontexte einzugehen, als es dort der Fall ist. Ihre Herangehensweise in „Automatic and Accurate Expansion of Abbreviations in Parameters“ [JLZZ18] ist, sich konkret auf Bezeichner in Parametern zu beschränken. Deren Auflösung wird in den entsprechenden abstrakten, beziehungsweise tatsächlichen Parametern und deren Typ gesucht. Hierzu werden zunächst die tatsächlichen Parameter aus dem Quelltext extrahiert, zusammen mit den entsprechenden abstrakten Parameter als Kontext für die Suche nach Auflösungskandidaten, oder umgekehrt, und zusammen mit den Typen dieser Parameter. Dann werden diese Parameter gespalten. Die Spaltung erfolgt anhand der Erkennung von Trennung durch Unterstrich oder Binnenmajuskel. Die so erhaltenen harten Wörter werden entweder, durch Abgleich mit einem englischen Wörterbuch, als natürlichsprachliche Wörter erkannt, oder es wird davon ausgegangen, dass es sich um eine Abkürzung handelt. Die so erkannten Abkürzungen werden anschließend über verschiedene Wissensquellen in priorisierter Reihenfolge aufgelöst. Für jede Wissensquelle wird eine andere Reihe an Heuristiken genutzt. Hierzu wird zunächst im Namen des zum Bezeichner gehörigen abstrakten, beziehungsweise tatsächlichen Parameters gesucht. Wird dort keine Auflösung gefunden, wird im Typ nach Auflösungskandidaten gesucht, dann erst in einem Wörterbuch. Wird in jedem dieser Schritte keine Auflösung gefunden, wird keine zurückgegeben. Dabei wurden folgende Heuristiken eingesetzt: Die Suche im Parameter-Namen des entsprechenden abstrakten, beziehungsweise tatsächlichen Parameters, wird in drei Schritten ausgeführt. Zunächst (1) wird überprüft, ob es sich bei der Abkürzung um ein Akronym des Parameter-Namens handelt. Ist dies erfolglos, (2) wird überprüft, ob die Abkürzung ein Präfix des Parameter-Namens oder eines seiner Segmente ist. Schließlich (3) wird überprüft, ob es sich bei der Abkürzung um eine Variante des Parameter-Namens handelt, in der Buchstaben weggelassen wurden. Letzteres wird dadurch erkannt, dass alle Buchstaben in der Abkürzung in der richtigen Reihenfolge, wenn auch eventuell mit Unterbrechungen, im Parameter-Namen vorkommen. Bei der Suche im Typ des Parameters wird zunächst der Name des Typs gespalten. Stimmt eine Abkürzung mit einem Token überein, oder ist sie ein Präfix davon, gilt dies als gefundene Auflösung. Falls beispielsweise eine Methode mehrfach aufgerufen wird, und der abstrakte Parameter aufgelöst werden soll, wird dieser Konflikt durch die Wahl des häufigsten Kandidaten entschieden. Außerdem können in solchen Fällen Abkürzungen rekursiv aufgelöst werden. Um ihren Ansatz zu evaluieren, verwenden die Autoren sieben verschiedene Probanden-Programme. Hierzu

wurden alle Abkürzungen aus jedem Programm gezählt. Hierbei wurde festgestellt, dass 36,74% aller Abkürzungen in Parametern zu finden sind. Zudem wurden 648 Abkürzungen aus Parametern manuell aufgelöst, um diese als Musterlösung für die Evaluation der Abkürzungsauflösung zu nutzen. Die Abkürzungsauflösung auf allen Programmen erzielt insgesamt eine Präzision von 95% und eine Ausbeute von 65%. Da das Verfahren aber per Definition nur auf diesen Kontext beschränkt eingesetzt werden kann, schlagen Jiang et al. für die Zukunft vor, es mit einem allgemeineren Verfahren in Kombination anzuwenden.

Newman et al. legen in „An Empirical Study of Abbreviations and Expansions in Software Artifacts“ [NDA+19] einen Überblick über einige bis zu diesem Zeitpunkt vorgestellte Verfahren dar. Sie vergleichen nicht nur die Güte der Abkürzungsauflösung von neun Verfahren, sondern stellen auch eine eigene Kategorisierung von Abkürzungstypen vor, vergleichen die Arbeiten bezüglich dieser einzelnen Abkürzungstypen, und untersuchen selbst die Eignung verschiedener Wissensquellen je Abkürzungstyp. Die Autoren teilen die Abkürzungen zunächst in zwei Kategorien ein: Einwort- und Multiwort-Abkürzungen. Diese werden jeweils wieder in zwei Typen unterteilt: Einwort-Abkürzungen können Präfixe oder Wörter mit weggelassenen Buchstaben sein; Multiwort-Abkürzungen können Akronyme oder Zusammensetzungen von Einzelwort-Abkürzungen sein. Sie durchsuchen fünf Programme welche in Java, C oder C++ geschrieben sind, mit Quelltextlängen zwischen neuntausend und viereinhalb Millionen Zeilen, nach den oben beschriebenen Abkürzungstypen. Von den Abkürzungstypen waren in jedem betrachteten Quelltext Präfixe die am häufigsten auftretenden, meistens gefolgt von Wörtern mit fehlenden Buchstaben, Akronymen und zuletzt zusammengesetzten Abkürzungen. Für die Analyse der Abkürzungstypen und Wissensquellen erzeugen die Autoren manuell eine Sammlung von Bezeichnern mit ihren Auflösungen. Außerdem werden Quelltext, Kommentare, Projektdokumentation, Sprachdokumentation, ein Informatik-Lexikon und ein Englisch-Wörterbuch als mögliche Wissensquellen für die Suche nach Auflösungskandidaten im Nachhinein nach den richtigen Auflösungen durchsucht, um festzustellen, welche dieser Quellen für die Kandidaten-Suche hilfreich sein können. Es werden auch verschiedene Stellen im Quelltext betrachtet, in denen Auflösungen zu finden sind. Allgemein sind Methoden hierbei die zuverlässigsten Quellen, wobei die Werte von Projekt zu Projekt stark schwanken. Beim Vergleich der einzelnen Quellen schneidet die Sprachdokumentation am besten ab, dicht gefolgt von der Projektdokumentation, wobei besonders in der Sprachdokumentation richtige Kandidaten vorkommen, welche ausschließlich dort zu finden sind. Besonders stark schwankt die Auswertung der Kommentare im Bezug auf die darin befindlichen Auflösungen. Auch wird gewertet wie oft nicht die Auflösungen, sondern die verschiedenen Abkürzungstypen an verschiedenen Stellen im Quelltext vorkommen. Für jeden Abkürzungstyp ist die Reihenfolge der Stellen im Quelltext, an denen er häufig auftritt, identisch. Allein die Ausprägung schwankt. So sind beispielsweise 80,3% aller Präfix-Abkürzungen und 37,6% aller Akronyme im Methodenrumpf zu finden. Für alle Abkürzungstypen ist der Methodenrumpf aber der Quelltext-Kontext, in welchem sie am häufigsten auftreten. Außerdem liegt, insbesondere bei der Sprachdokumentation, ein Großteil der richtigen Auflösungen nicht adjazent vor, die einzelnen Teile eines zusammengesetzten Bezeichners liegen also nicht in der gleichen Reihenfolge vor, wie in der Abkürzung. All die oben beschriebenen Analysen werden zudem verwendet, um zu evaluieren, ob die Strategien verschiedener Arbeiten zur Problematik der automatischen Abkürzungsauflösung angemessen sind, im Hinblick auf die Realität von Abkürzungen in Quelltext und deren Auflösungen in verschiedenen Wissensquellen. Newman et al. stellen fest, dass keines der getesteten Verfahren für den Fall nicht adjazent auftretender Auflösungen eine explizite Strategie hat. Allerdings suchen einige Verfahren die Abkürzungen für die einzelnen Teilbezeichner nach der Spaltung weitgehend unabhängig voneinander, was diesen Fall implizit abdeckt. Die getesteten Verfahren sind die von Lawrie et al. aus den Jahren 2007 [LFB07] und 2011 [LB11], *AMAP* [HFB+08], *LINSEN* [CDMM12], das Verfahren von Alatawi et al. aus dem

Jahre 2018 [AXY18], *TIDIER* [GPAG13], das Verfahren von Jiang et al. aus dem Jahre 2018 [JLZZ18], *Lingua::IdSplitter* [CAHV15] und *TRIS* [GGG+12]. Nicht alle Verfahren nutzen die gleichen Metriken zu ihrer Evaluation, was Newman et al. bedauern. Ebenso gehen nicht alle Verfahren auf einzelne Abkürzungstypen. Es lassen sich also nicht alle Verfahren direkt quantitativ miteinander vergleichen, allerdings gibt es einige Besonderheiten, welche hervorgehoben werden können: Unter den Verfahren, welche die allgemeine Genauigkeit angaben, schnitt *TRIS* mit 86% am besten ab - die Evaluation von *TRIS* [GGG+12] beschreibt allerdings nur eine Genauigkeit der Spaltung, welche, da die Spaltentscheidung bei *TRIS* anhand der Auflösung gefällt wird, als kombinierte Genauigkeit für Spaltung und Auflösung angesehen wird. Abgesehen von *TRIS* war die höchste Genauigkeit die von Alatawi et al. mit 78%. *AMAP* und *LINSEN* schlüsseln ihre Genauigkeit nach den oben beschriebenen Abkürzungstypen auf und sind beide bei Präfixen am präzisesten, dicht gefolgt von Wörtern mit fehlenden Buchstaben. *TRIS*, *Lingua::IdSplitter* und Jiang et al. geben außerdem Präzision und Ausbeute an, wobei Jiang et al. und *TRIS* sich die größte Präzision von 95% teilen, während *TRIS* mit 91% die größte Ausbeute aufweist, und wichtig zu erwähnen ist, dass Jiang et al. nur Abkürzungen in Parametern betrachten. Außerdem wurde evaluiert welche Wissensquellen welches Verfahren einsetzt: alle Verfahren nutzen den Quelltext, fast alle nutzten Wörterbücher, wenn auch oftmals in letzter Instanz. Außer *TRIS* nutzt nur *TIDIER* die Sprachdokumentation.

4.2 Aufspalten von Bezeichnern in harte und weiche Wörter

Feild et al. führen in ihrer Arbeit „An Empirical Comparison of Techniques for Extracting Concept Abbreviations from Identifiers“ [FBL06] einen empirischen Vergleich zwischen einem gierigen Algorithmus, einem zufälligen Algorithmus und mehreren neuronalen Netzen durch, welche alle zum Ziel haben, harte Wörter, beziehungsweise Multiwort-Bezeichner ohne Trennzeichen, in ihre korrekten weichen Wörter aufzuspalten. Jedes der neuronalen Netze hat die gleiche Form, sie werden aber auf unterschiedlichen Datensätzen trainiert. Dazu legen vier Entwickler jeweils eine Musterlösung an, in welchem sie jeweils 1.000 zufällig ausgewählte Bezeichner, aus 186 Programmen mit insgesamt 746.345 Bezeichnern, manuell auf Auflösungen abbilden. In diesen Lösungs-Bezeichnern sind zwischen den weichen Wörter Trennzeichen hinzugefügt. Aus jedem dieser Datensätze wurde ein Trainingsdatensatz extrahiert, und der Rest als Testdatensatz behandelt. Je ein neuronales Netz wird auf je einem dieser Trainingsdatensätze trainiert, eines auf allen vier. Dann wird jedes dieser neuronalen Netze auf jedem Testdatensatz, inklusive des kombinierten Testdatensatzes, getestet. Der gierige Algorithmus und ein Zufallsalgorithmus wurden ebenfalls auf all diesen Testdatensätzen getestet. Der zufällige Algorithmus hat die gleiche Frequenz an Spaltungen wie die Musterlösung. Das auf allen Trainingsdatensätzen trainierte neuronale Netz ist immer besser als der gierige Algorithmus. Auf dem kombinierten Testdatensatz erzielt dieses neuronale Netz eine Genauigkeit von 78,12%, der gierige Algorithmus spaltet 75,69% der Bezeichner richtig. Jedes neuronale Netz ist auf dem Testdatensatz am besten, der aus der gleichen Quelle stammt wie sein Trainingsdatensatz. So erzielte eines der Netze eine Genauigkeit von 95,14%. Bei Anwendung eines neuronalen Netzes auf einen anderen Datensatz fällt die Genauigkeit allerdings manchmal unter die des gierigen Algorithmus. Auf einem der Testdatensätze ist sogar der Zufallsalgorithmus besser als der gierige Algorithmus. Letzteres wird darauf zurückgeführt, dass die Musterlösung hier insgesamt weniger Spaltungen vorsieht, und der Zufallsalgorithmus weit konservativer spaltet als der gierige Algorithmus. Laut den Autoren deutet die Empfindlichkeit der neuronalen Netze gegenüber der Quelle ihres Datensatzes darauf hin, dass die Entwickler, welche ihre Entscheidungen basierend auf einem intuitiven Verständnis der Bezeichner getroffen haben, sich bezüglich der Frage, wo gespalten werden müsse und wo nicht, uneinig waren. Und obwohl das auf allen Quellen trainierte neuronale Netz immer besser abschnitt als der

gierige Algorithmus, übertraf dieser Unterschied selten fünf Prozentpunkte.

Enslen et al. stellen in „Mining Source Code to Automatically Split Identifiers for Software Analysis“ [EHPVS09] ihr Verfahren namens *Samurai* vor. Sie betonen, dass die meisten Software-Analyse-Werkzeuge, welche versuchen, Information aus Multiwort-Bezeichnern zu gewinnen, sich auf Quelltext-Konventionen wie Binnenmajuskelschreibweise oder Trennung mit Unterstrich stützen. Enslen et al. heben die Problematik hervor, dass Quelltext-Konventionen wie die Binnenmajuskelschreibweise nicht immer eingehalten werden, was auch gute Gründe für die Leserlichkeit haben kann (zum Beispiel: „DAYSforMONTH“). *Samurai* selbst basiert auf der Wortfrequenz im Quelltext, und braucht daher kein vordefiniertes Wörterbuch. Anhand der Frequenz eines Wortes innerhalb des betrachteten Programms und in einer Menge mehrerer Programme wird eine Formel aufgestellt, die einem Wort eine Wertung zuweist. Das Verfahren spaltet Bezeichner in zwei Schritten. Im Ersten wird die Binnenmajuskelschreibweise berücksichtigt, sowie auf andere Verwendungen von Großbuchstaben in Bezeichnern eingegangen. Diesen Schritt nennen Enslen et al. *Spaltung mit gemischter Kapitalisierung*. Hier wird die Wort-Wertung, welche aus der Frequenz-Analyse gewonnen wurde, genutzt, um zu entscheiden, ob Großbuchstaben als Binnenmajuskel oder als vollständig groß geschriebenes Teilwort zu verstehen sind, und somit der Bezeichner in seine harten Wörter aufgeteilt. Im zweiten Schritt werden aus den so entstanden harten Wörtern die weichen Wörter gewonnen. Hierzu werden die harten Wörter an jeder erdenklichen Stelle in zwei Hälften gespalten, und diese Spaltung sowohl anhand der eben beschriebenen Bewertungsfunktion bewertet, als auch sichergestellt, dass keine der Hälften ein übliches Präfix oder Suffix ist, da sonst angenommen würde, dass diese Hälften zusammen gehören. Ist die normierte Bewertung aller so entstandenen potentiellen Spaltungen schlechter als das harte Wort, wird die Spaltung verworfen. Diese Etappe wird *Spaltung bei einheitlicher Buchstabengröße* genannt und wird bei Bedarf, welcher wiederum von der Wertung der weichen Wörter abhängt, rekursiv wiederholt. Bezüglich der Wertungsfunktion ist hervorzuheben, dass gerade bei kleinen Programmen die Wortfrequenz-Informationen aus dem Programm selbst nicht ausreichend sind, um ein Urteil zu fällen, sondern dies erst mit Hilfe der Daten aus einer Menge an Programmen zuverlässig möglich wird. Die Auswertung wurde nach Verfahren, Syntax der Bezeichner und Teilwörter, sowie nach der der Frage, ob Bezeichner und Teilwörter hätten gespalten werden sollen, aufgeschlüsselt. *Samurai* wurde auf einer Musterlösung aus 8.466 Bezeichnern, welche zufällig aus 9.000 Open-Source Projekten gewonnen wurden, evaluiert. Dieser Datensatz beinhaltet auch Bezeichner, die nicht gespalten werden sollten. Dabei erzielte das Verfahren eine Genauigkeit von 97%.

4.3 Domänenfremde verwandte Arbeiten

Die hier beschriebenen Arbeiten befassen sich mit dem gleichen Problem, der Auflösung von Abkürzungen, in einer anderen Domäne als in Programm-Quelltext.

Ratinov und Gudes befassen sich in „Abbreviation Expansion in Schema Matching and Web Integration“ [RG04] damit, verteilte Daten aufeinander abzubilden. Zum Beispiel um Daten aus verschiedenen Datenbanken zu verknüpfen, um Konsistenz und Prägnanz der Daten gewährleisten zu können. Da auch in diesem Kontext oft Abkürzungen verwendet werden, befassen sie sich hierzu mit deren Auflösung. Dies birgt bei Datenbanken die besondere Schwierigkeit, dass es selten eine Dokumentation oder Ähnliches gibt, worin über Abkürzungen aufgeklärt würde. Die Autoren stellen zwei Verfahren vor: (1) Einen probabilistischen Algorithmus, der eine Heuristik befolgt, und (2) ein neuronales Netz. Beide verwenden Abkürzungsmuster um zu evaluieren, ob ein Begriff eine plausible Auflösung darstellt. Mit diesen Mustern ist eine Markierung der Buchstaben im Begriff gemeint, welche in der Abkürzung auftreten. Zum Beispiel kann die Abkürzung „itr“ aus dem Wort

„iterator“ sowohl mit dem Muster „VCxCxxxx“ als auch dem Muster „VCxxxxxC“ oder „VxxxxCxC“ gewonnen werden, wobei „x“ für nicht genutzt, „C“ für Konsonant und „V“ für Vokal steht. Die Trainingsdaten für das neuronale Netz werden mit einem probabilistischen Werkzeug generiert, welches vergleichbar mit dem heuristischen funktioniert. Sowohl der heuristische Algorithmus, als auch das neuronale Netz, erhalten zur Suche nach Auflösungskandidaten Zugriff auf eine Wissensquelle. Für die Evaluation wurde als Wissensquelle für die Kandidatensuche der „Brown Corpus“ [FK79] verwendet, welcher aus einer Million Wörter aus verschiedenen Quellen besteht. Die Richtigkeit wurde manuell nach Ausführung der verglichenen Werkzeuge geprüft, in dem die Ergebnisse verglichen und eins, beide oder keins für richtig befunden wurde, wobei das neuronale Netz oftmals mehrere Hundert Kandidaten für eine Abkürzung anbot. Insgesamt hatte das neuronale Netz eine etwas höhere Ausbeute als die Heuristik, bei weit geringerer Präzision. Die Ausbeute wurde allerdings für wichtiger befunden, da Ratinov und Gudes planen in Zukunft, in einem weiteren Schritt, ein Werkzeug zur Mehrdeutigkeitsauflösung zu verwenden.

Zhang et al. stellen in „Entity Linking with Effective Acronym Expansion, Instance Selection and Topic Modeling“ [ZSST11] ein Verfahren zur „Entitäts-Verknüpfung“ vor, welches das Abbilden von Bezeichnern in einem Dokument, auf eine Wissensdatenbank oder Enzyklopädie, erleichtern soll. Hierzu sollen Akronyme automatisch zu ihren ausführlichen Formulierungen aufgelöst werden, wobei ein überwachter Lernalgorithmus dazu verhelfen soll, kompliziertere Akronyme aufzulösen. Als Wissensdatenbank wurde hier Wikipedia herangezogen. Das Problem der Verlinkung von Bezeichnern zu den entsprechenden Einträgen der Wissensdatenbank beinhaltet das Auflösen von mehreren Varianten eines Namens oder mehreren Bedeutungen eines Bezeichners. Unter anderem kann dieser aufgelöst werden, indem davon ausgegangen wird, dass bei Mehrdeutigkeit eher mehrfach die gleiche Entität in einem Dokument gemeint ist, als verschiedene. Da die Wikipedia-eigenen Begriffserklärungen zu Rate gezogen werden, lassen sich einige Akronyme schon durch diese auflösen. Akronyme werden im Text gefunden, indem davon ausgegangen wird, dass sie vollständig groß geschriebene Wörter sind. Auflösungskandidaten werden zunächst gesucht, indem geprüft wird, ob nach dem Akronym ein Begriff in Klammern steht, oder ob das Akronym in Klammern steht und ein Begriff davor. Dabei werden gegebenenfalls mehrere Kandidaten generiert, indem Stoppwörter entweder mit einbezogen oder herausgekürzt werden. Weitere Kandidaten werden bestimmt, indem alle Token im selben Dokument gesucht werden, welche den gleichen Anfangsbuchstaben haben. Anschließend wird jedes solche Token, zusammen mit darauf folgenden Token, bis entweder ein Satzzeichen oder mehr als zwei Stoppwörter dazwischen liegen, als ein Begriffs-Kandidat betrachtet. Aus diesem Kandidaten werden wiederum, basierend auf Stoppwörtern, die mit einbezogen oder weggelassen werden, mehrere Kandidaten generiert. Um aus dieser Menge an Kandidaten nur valide Auflösungen zu betrachten, wird ein überwachter Lern-Algorithmus eingesetzt, konkreter eine Stützvektormaschine. Dabei handelt es sich um einen maschinell lernenden Klassifizierer. Die möglichen Klassen, denen die Abkürzung zugewiesen werden kann, sind die Auflösungskandidaten. Diese Entscheidung wird basierend auf gewissen Eigenschaften von Abkürzung und Kandidat getroffen. Die Eigenschaften, welche hier als Merkmale dienen, sind Merkmale wie die Länge des Akronyms, die längste aneinander hängende Übereinstimmung, oder der Unterschied der Länge von Akronym und Auflösungskandidat, aber auch sprachorientierte Merkmale wie die Zahl der Verben im Auflösungskandidaten, oder speziellere Merkmale wie die Exponentialfunktion des Abstands der Abkürzung vom Auflösungskandidaten im Text. Jedes Akronym-Auflösungskandidaten-Paar wird dem Stützvektormaschinen-Klassifizierer präsentiert, welcher ihnen eine Punktzahl zuweist. Gibt es keine positiv bewerteten Kandidaten, wird das Akronym nicht aufgelöst. Ansonsten wird das Ergebnis mit der besten Wertung ausgewählt. Ein Trainingsdatensatz für die Stützvektormaschine wird automatisch generiert, indem Bezeichner, welche eigentlich in ihrer eindeutigsten Formulierung vorliegen, durch Synonyme ersetzt werden.

So wurden aus 1,7 Millionen Dokumenten 45.000 Instanzen durch mehrdeutige Synonyme substituiert. Ein weiterer Trainingsdatensatz für die Evaluation des Akronym-Expandierers wurde halbautomatisch aus der Dokumentensammlung des „Knowledge Base Population Task“ der „Text Analysis Conference (TAC) 2010“ [JGDG10] erzeugt. Es wurden automatisch Bezeichner gesucht, welche Akronyme sein könnten, und diese manuell durch Wörter aus dem selben Dokument aufgelöst. Wurde keine Auflösung gefunden, wurde davon ausgegangen, dass es keine gibt (NIL). Das Verfahren wurde, bezüglich seiner Genauigkeit, mit zwei anderen verglichen, und die Ergebnisse außerdem danach aufgeschlüsselt, ob die Abkürzung eine Auflösung besitzt, und wenn ja ob die Token, welche den Begriff zusammensetzen, in der gleichen Reihenfolge in diesem wie in der Abkürzung vorkommen. Das Verfahren erzielte in der Evaluation eine Auflösungsgenauigkeit von 92,9%. Die Genauigkeit wurde auch unter Nutzung oder Nichtnutzung verschiedener Merkmale evaluiert. Das oben genannte beste Ergebnis wurde unter Einfluss aller Merkmale erzielt, wobei die Autoren besonders hervorheben, dass die von ihnen vorgeschlagenen semantischen Merkmale demnach eine Verbesserung hervorgebracht haben. Außerdem wurden die Ergebnisse des Verfahrens verglichen, je nach dem welcher Trainingsdatensatz genutzt wurde: Der halbautomatisch erzeugte, oder der automatisch erzeugte Datensatz. Mit dem halbautomatisch erzeugten Trainingsdatensatz war das Verfahren weniger genau, mit diesem wurden 82,8% der Bezeichner auf den richtigen Eintrag der Wissensdatenbank verlinkt. Mit dem automatisch erzeugten Trainingsdatensatz waren es 83,6%. Wurden beide Trainingsdatensätze zusammen genutzt, fiel die Genauigkeit auf 81,2%.

Liu et al. befassen sich in „Exploiting Task-Oriented Resources to Learn Word Embeddings for Clinical Abbreviation Expansion“ [LGM+15] mit der Frage der Abkürzungsauflösung im medizinischen Bereich, konkreter in der Intensiv-Pflege. Hier besteht oftmals das Problem, dass weder viel Zeit zum Aufschreiben, noch zum Lesen von Notizen besteht. Außerdem gibt es für viele Begriffe keine allgemeinen Abkürzungen oder Abkürzungsregeln. Dies verstärkt außerdem das allgemeine Problem, dass Abkürzungen oft uneindeutig sind und in Abhängigkeit vom Kontext verschiedenes meinen. In diesem Ansatz wurden zunächst, mittels regulärer Ausdrücke, alle Abkürzungen aus einer Notiz einer Intensiv-Pflege-Fachkraft entnommen, und dann versucht alle Auflösungskandidaten aus einer Sammlung medizinischer Abkürzungen auf *allacronyms.com* [Rad15] zu ermitteln. Es wurden verschiedene Quellen wie Wikipedia, wissenschaftliche Artikel, Bücher und weitere Texte über Intensiv-Pflege zu Rate gezogen, um Worteinbettungen zu trainieren. Laut eigener Aussage sind diese Autoren die ersten, welche Worteinbettungen für diese Aufgabe verwenden. Diese Worteinbettungen wurden genutzt um die Semantik der Abkürzungen und ihrer Kandidaten zu vergleichen. Für Multiwort-Bezeichner werden die Einbettungen aller Token summiert. Die semantische Übereinstimmung ist außerdem wahrscheinlicher, wenn Abkürzung und Kandidat in ähnlichen Kontexten vorliegen. Die Autoren gehen davon aus, dass manchmal auch kontextunabhängige Information hilfreich ist. So wird auch die allgemeine Verbreitung einer Abkürzung für einen bestimmten Begriff in die Bewertung des entsprechenden Kandidaten einbezogen. Insgesamt stellen sie also eine Bewertungsfunktion auf, welche sich aus der allgemeinen Häufigkeit und der durch die Einbettung bestimmte Kontext-Nähe der Abkürzung zum Auflösungskandidaten zusammensetzt, welche durch eine Konstante gegeneinander gewichtet werden. Für die Evaluation der Genauigkeit des Verfahrens wurden 1.160 Protokolle von Intensiv-Pflege-Zulassungsanträgen ausgewertet. Es wurde *word2vec* [MCCD13] verwendet, um die Wort-Einbettungen auf 42.506 Wikipedia-Artikeln zu trainieren. Es wurde eine Musterlösung aus 100 Protokollen durch einen der Autoren der Arbeit manuell aufgelöst. Das Verfahren wurde gegen das einfache Auswählen des am besten bewerteten Kandidaten aus *allacronyms.com* oder die Verwendung von Wort-Einbettungen auf anderen Trainingsdaten verglichen. Das Verfahren erzielte eine Genauigkeit von 82,27%. Außerdem hat eine der Autorinnen die Fähigkeit von verschiedenen Gruppen von Domänen-Experten geschätzt und kam zu dem Ergebnis, dass das Verfah-

ren Leistungen erzielt die nahe an Domänen-Experten herankommen. Dass das Verfahren nicht noch besser war wird darauf zurück geführt, dass nicht alle Abkürzungen in der Wissensquelle vorkommen (was 27% der Fehler des Verfahrens ausmacht), dass die Trainingsdaten nicht für jeden Auflösungskandidaten über genügend Beispiele verfügten (54% der Fehler), und dass einige Abkürzungen zu kompliziert formuliert seien (19% der Fehler). Das Verfahren von Liu et al. erzielt diese Leistung, ohne große Mengen gekennzeichneter Trainingsdaten zu benötigen, und das Experiment demonstriert, dass die Verwendung von aufgabenorientierten Ressourcen weit effektiver ist als die einer allgemeineren Sammlung.

5 Analyse

Softwareentwickler schreiben Quelltext in erster Linie, um Anweisungen und Verhaltensvorgaben zu formulieren, welche der Computer erfüllen soll. Doch nicht nur der Autor des Quelltextes sollte dazu in der Lage sein den Quelltext als Text zu verstehen. Sowohl andere menschliche Entwickler, als auch automatische Informationsrückgewinnungswerkzeuge, sind darauf angewiesen, den Quelltext als Text zu verstehen und zu interpretieren, um das Programm zu warten, zu erweitern, Quelltext wiederzuverwenden oder um die Erfüllung von Anforderungen zu überprüfen. Leider wird der Quelltext dieser notwendigen Allgemeinverständlichkeit nicht immer gerecht. So werden Quelltextbezeichner oft nicht als vollständige natürliche Begriffe ausgeschrieben, sondern abgekürzt. Zwar bedienen sich Entwickler hierbei allgemein noch der natürlichen Sprache, schließlich sind sie darauf angewiesen ihren eigenen Quelltext zumindest während der Implementierung selbst verstehen zu können, doch ohne das Hintergrundwissen des Autors sind solche Abkürzungen oft unverständlich oder wenigstens uneindeutig. Die Auflösung solcher Abkürzungen zu den natürlichsprachlichen Begriffen, für welche sie stehen, dient der Rückgewinnung der allgemeinen Verständlichkeit des Quelltextes.

5.1 Herausforderungen

Die Auflösung von abgekürzten Quelltextbezeichnern in einem Programm, zu den entsprechenden natürlichsprachlichen Begriffen, lässt sich in drei Schritte aufteilen: (1) Das Erkennen von Abkürzungen, (2) das Aufspalten der Bezeichner zu mehreren Token, und (3) die Auflösung der Abkürzungen. Diese drei Schritte müssen nicht zwangsläufig strikt getrennt hintereinander ausgeführt werden, und doch lassen sie sich allgemein auf die Auflösungen von Quelltextabkürzungen anwenden. Um Abkürzungen aufzulösen, ist es offensichtlich notwendig, eine Abkürzung als solche zu erkennen. Dies kann einerseits vor der Auflösung passieren, sodass nur auf Bezeichner, welche explizit als abgekürzt erkannt wurden, das Auflösungsverfahren angewendet wird. Andererseits kann ein Verfahren zur Auflösung von Abkürzungen auch auf alle Bezeichner angewandt werden, um dann implizit davon auszugehen, dass Bezeichner, für welche keine Auflösung ausgewählt wurde, nicht abgekürzt sind und somit keiner Auflösung bedürfen. Die Frage der impliziten Erkennung von Abkürzungen nach der Auflösung hängt vom jeweiligen Auflösungsverfahren ab. Es könnte sein, dass keine gefunden wurde. In diesem Fall kann nun implizit davon ausgegangen werden, dass der Bezeichner nicht abgekürzt ist und keiner Auflösung bedarf, oder er kann grundsätzlich als ein Fehlverhalten des Auflösungsverfahrens gewertet werden, oder

es kann nachträglich eine Methode zur expliziten Erkennung von Abkürzungen eingesetzt werden. Es könnte auch sein, dass die gefundenen Auflösungskandidaten einem bestimmten Qualitätsmaß nicht entsprechen, welches das Auflösungsverfahren für die Bewertung der eigenen Güte, oder zur Auswahl des richtigen Kandidaten verwendet. Die Konsequenz ist letztendlich die gleiche, wie wenn keine Auflösung gefunden wurde. Es könnte zuletzt auch sein, dass der Bezeichner selbst, beziehungsweise nicht aufgelöste Teilbegriffe aus diesem, als seine Auflösung betrachtet werden. Wie dies zu bewerten ist, hängt vom jeweiligen Auflösungsverfahren ab. Es könnte einerseits sein, dass das Auflösungsverfahren diese Auflösung zu sich selbst als den Standard betrachtet, wenn keine andere Auflösung gefunden wurde. Dann wäre die Konsequenz die gleiche wie wenn keine oder keine ausreichend gute Auflösung gefunden wurde, oder aber das Auflösungsverfahren kommt tatsächlich zu dem Schluss, dass der Bezeichner selbst seine richtige Auflösung ist. Das wäre wiederum eine explizite Erkennung einer Nichtabkürzung. Es kann allerdings auch wünschenswert sein, Abkürzungen beziehungsweise Nichtabkürzungen explizit vorher zu erkennen. Dies kann einerseits dazu dienen, keine falschen Auflösungen zu erzeugen, und andererseits Rechenaufwand einzusparen, indem Nichtabkürzungen vom Verfahren von vorne herein ignoriert werden können. Eine auf der Hand liegende Methode zum Erkennen von Nichtabkürzungen ist der Abgleich mit einem Wörterbuch. Kommt der Bezeichner in einem Wörterbuch vor, wird davon ausgegangen, dass es sich nicht um eine Abkürzung handelt, und er braucht nicht aufgelöst werden. Doch ein einfacher Abgleich des Bezeichners mit einem Wörterbuch liefert nicht zwangsläufig die richtige Antwort. Nicht jeder Bezeichner oder jedes Token, welches keinen natürlichsprachlichen Begriff darstellt, ist abgekürzt. Beispielsweise kann ein Bezeichner zufällig einem natürlichsprachlichen Begriff entsprechen, obwohl er als Abkürzung eines anderen gemeint ist, wie Beispiel 5.1 zeigt. Außerdem kann ein Quelltextbezeichner aus mehreren nicht abgekürzten Wörtern zusammengesetzt sein. Für diesen Fall kann es also entweder Sinn ergeben, den Bezeichner vor der Erkennung von Abkürzungen zu spalten, oder aber die Erkennung für die Spaltung zu nutzen.

Beispiel 5.1: Beispiel eines abgekürzten Bezeichners, welcher als natürliches Wort lesbar ist

INDIRECT als Abkürzung für ***IN****tent-****D****r****I****ven* ***RE****quirements-to-****C****ode* ***T****raceability.*

Umgekehrt könnte auch versucht werden, Abkürzungen explizit zu erkennen. Dies wirft die Frage auf, wie eine Abkürzung aussieht. Gibt es verallgemeinerbare Regeln für den Aufbau einer Abkürzung? Gibt es Untergruppen von Abkürzungen mit solchen Regeln? Für Abkürzungen im allgemeinen scheint es diese nicht zu geben. Abkürzungen können ein einzelner Buchstabe oder eine Wortneuschöpfung sein. Zwar ist es nahezu immer der Fall, dass eine Abkürzung den gleichen Anfangsbuchstaben hat wie ihre Auflösung, und für bestimmte Abkürzungsstrategien sind weitere solcher Regeln festzustellen, doch lässt sich diese Information erst nutzen, wenn die Abkürzung und ein Auflösungskandidat vorliegen, nicht zur Erkennung der Abkürzung an sich. Dennoch lässt sich eben der Gedankengang solcher Abkürzungsstrategien zurück verfolgen, um von Abkürzungen auf ihren natürlichsprachlichen Ursprung zu schließen. Daraus lassen sich verschiedene Typen von Abkürzungen ableiten, die wiederum verschiedene Strategien zu ihrer Auflösung begünstigen.

Wenn von abgekürzten Quelltextbezeichnern, beziehungsweise Abkürzungen in Quelltextbezeichnern die Rede ist, muss auch darauf eingegangen werden, dass Quelltextbezeichner nicht zwangsläufig einen einzelnen unteilbaren Term darstellen. Quelltextbezeichner bestehen mitunter aus mehreren Token, welche gemeinsam einen Begriff ergeben. Nun können einzelne, mehrere oder alle Token eines Bezeichners Abkürzungen sein. Diese bilden jeweils auf einen natürlichen Term ab. Um diese Abkürzungen auflösen zu können, müssen

die Grenzen dieser Token erkannt werden. Ansonsten könnte wohl kaum aus der übrigen sprachlichen Information eines Quelltextbezeichners nachvollzogen werden, welche natürlichen Wörter auf welche Art und Weise abgekürzt worden sind, oder überhaupt wie viele natürliche Wörter die richtige Auflösung des Bezeichners enthält. Dennoch ist auch dieser Schritt nicht zwingend strikt nach der Erkennung oder vor der Auflösung auszuführen. Die Token, oder zumindest einige der Token, könnten auch durch das Erkennen nicht abgekürzter Terme im Bezeichner erkannt werden, wie in Beispiel 5.2. Die Spaltung eines Bezeichners kann auch nach der Erzeugung mehrerer Auflösungskandidaten entschieden werden, indem mehrere, wenn nicht alle, möglichen Spaltungen in Token als Ausgangspunkt für die Suche nach der richtigen Auflösung in Erwägung gezogen werden, wie in Beispiel 5.3.

Beispiel 5.2: Beispiel eines natürlichen Wortes in einem zusammengesetzten Bezeichner

fface als Abkürzung für *flag face*.

Beispiel 5.3: Beispiel eines Bezeichners, der je nach Auflösung anders gespalten wird

strcmp kann zu *str, cmp* für die Auflösung *string compare* oder zu *strc, mp* für die Auflösung *strict map* gespalten werden.

Dies wirft die Frage nach dem dritten und letzten Schritt der Abkürzungsauflösung auf: Die Auflösung einer Abkürzung zu einem natürlichsprachlichen Begriff. Hier müssen die, im Quelltext und anderen Artefakten, verfügbaren Hinweise genutzt werden, um auf die richtige Auflösung schließen zu können. Was über die Abkürzung bekannt ist, ist sie selbst und ihr Kontext. Da eine Abkürzung auf die eine oder andere Art eine Verkürzung des ursprünglichen natürlichen Begriffs ist, enthalten die Buchstaben der Abkürzung und ihre Anordnung Hinweise auf die Auflösung. Dies verdeutlicht auch den Zusammenhang zwischen der Erkennung und der Auflösung von Abkürzungen, da für beides die gleichen Merkmale genutzt werden können. Informationen aus dem Kontext können insofern zur Auflösung beitragen, dass Wörter oft in manchen Zusammenhängen wahrscheinlicher sind als in anderen. Hier wird auch die Bedeutung und Abhängigkeit der Spaltung deutlich. Die in der Spaltung aus dem gleichen Bezeichner heraus gelösten Token sind der unmittelbarste Kontext einer Abkürzung. Zugleich können verschiedene Spaltungskandidaten und verschiedene daraus hervorgehende Auflösungskandidaten auf unterschiedliche Kontexte schließen lassen. Welcher Spaltungskandidat der wahrscheinlichste ist, hängt also auch davon ab, welche Auflösungen von Abkürzungen er ermöglicht, welche Kontexte sich daraus ergeben und wie wahrscheinlich diese Auflösungen jeweils in diesem Kontext sind, wie Beispiel 5.4 zeigt. Weitere Hinweise auf die Auflösung, beziehungsweise Auflösungskandidaten, sind in Artefakten wie der Dokumentation, Fachliteratur oder Wörterbüchern zu finden. Dass ein Wort in einem Wörterbuch vorkommt, macht es allgemein plausibel, dass es in Sprache, also auch in Quelltext, verwendet wird. Ein Auflösungskandidat, welcher in der Dokumentation der Software oder Fachliteratur der gleichen Domäne zu finden ist, ist natürlich um so plausibler, da diese Artefakte und der Quelltext zusammenhängende Begebenheiten beschreiben.

Tabelle 5.1: Auftreten der verschiedenen Abkürzungstypen in Quelltext allgemein, laut Newman et al. [NDA+19]

Präfix	Weggelassene Buchstaben	Akronyme	Multiwort-Abkürzungen
55,3%	23,9%	20,8%	0,0%

Anteil aller Abkürzungstypen an der Gesamtzahl der Abkürzungen im Quelltext.

Beispiel 5.4: Beispiel eines Bezeichners, der mit Hilfe des Kontextes aufgelöst werden kann

Die Deklaration *String str_name;* der Variable, die den Bezeichner *str_name* trägt, lässt die im Bezeichner vorhandene Abkürzung *str* über zwei Wege aus dem Kontext auflösen: erstens steht die Auflösung *String* im Datentyp der Deklaration, zweitens kann bei Spaltung am Unterstrich das Wort *name* aus dem Bezeichner entnommen werden, was semantisch wieder den Rückschluss auf den Datentyp *String* erlaubt und somit auf diesen Auflösungskandidaten hinweist, beziehungsweise die Entscheidung *String* als die richtige Auflösung anzusehen bestärkt.

5.2 Erkennen von Abkürzungen

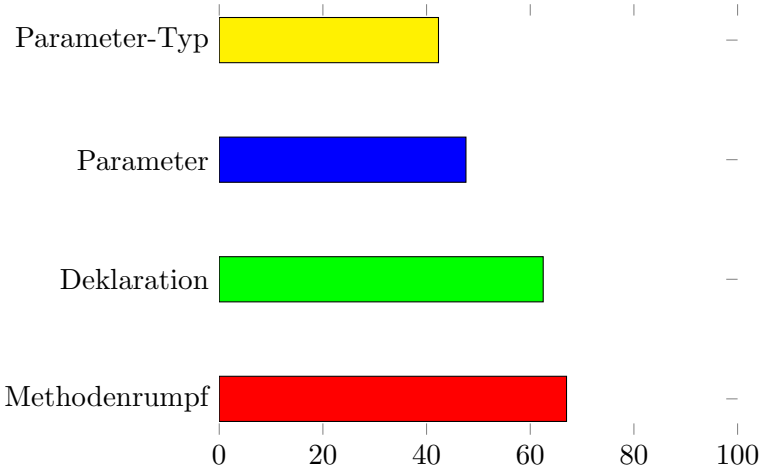

Abbildung 5.1: Anteil aller Abkürzungen in %, welche an gewissen Orten im Quelltext auftreten. Die Summe ergibt nicht 100%, da einige Abkürzungen an mehreren Stellen auftreten.

Bevor Abkürzungen aufgelöst werden können, muss als erstes festgestellt werden, was eine Abkürzung ist, beziehungsweise ob ein Bezeichner eine Abkürzung ist oder eine Abkürzung enthält. Entwickler verwenden Abkürzungen in Bezeichnern, welche sie schreiben. Das heißt also in den Klassen, Methoden und Variablen, welche der Entwickler definiert. Zudem sind Abkürzungen innerhalb des Quelltextes nicht unbedingt gleichmäßig verteilt: Laut Newman et al. [NDA+19] treten Abkürzungen am häufigsten im Methodenrumpf auf (67,0% aller Abkürzungen), gefolgt von den Namen von Deklarationen (62,5% aller Abkürzungen), während sie in Parametern (47,6% aller Abkürzungen) beziehungsweise deren Typ (42,3% aller Abkürzungen) am seltensten sind. Es wird in Abschnitt 5.1 bereits erörtert, dass Abkürzungen grundlegend auf natürlichsprachlichen Begriffen basieren. Bezeichner dienen dazu, Bestandteile des Programms zu bezeichnen, also gewissermaßen

dessen Funktionsweise zu beschreiben. Die Begriffe die demnach in den Bezeichnern, abgekürzt oder nicht abgekürzt, vorkommen, entspringen demnach der Domäne, in welcher das Programm eingesetzt werden soll, oder der Programmiersprache selbst. Dabei ist auch zu beachten, dass der Entwickler nicht zwangsläufig alle Abkürzungen selbst erdacht hat. Einige Abkürzungen mögen in der Domäne, oder in Quelltext allgemein, auch außerhalb dieses Programms üblich sein. Solche allgemein üblichen Abkürzungen, wie „cp“ für „copy“ zu beachten kann hilfreich sein, doch sollten auch die Grenzen der Domäne beachtet werden, also sollte die Abkürzung „rpm“ beispielsweise im Quelltext des *RPM Package Manager* nicht für „revolutions per minute“ gehalten werden. Nun gibt es verschiedene Wege wie ein Begriff abgekürzt werden kann. In dieser Arbeit ist in diesem Zusammenhang von Abkürzungstypen die Rede. Hier wird zwischen vier Abkürzungstypen unterschieden: (1) Präfix-Abkürzungen sind, wie der Name schon sagt, Abkürzungen die aus dem Anfang des Wortes bestehen, beziehungsweise das Wort, in welchem ein beliebig langes Ende weggelassen wurde, wie Beispiel 5.5 zeigt. Einige Abkürzungen sind auch (2) Wörter, aus welchen Buchstaben an beliebiger Stelle entfernt wurden. Allerdings wird nahezu immer der erste Buchstabe behalten, und Vokale scheinen öfter entfernt zu werden als Konsonanten, wie in Beispiel 5.6. Hinzu kommen Bezeichner, die aus mehreren Wörtern oder Abkürzungen zusammengesetzt sind. So auch (3) Akronyme, welche aus jedem Wort nur den ersten Buchstaben enthalten, siehe Beispiel 5.7, und (4) Zusammensetzungen aller zuvor beschriebenen Abkürzungstypen, wie in Beispiel 5.8. Akronyme hingegen stellen meistens einen Begriff dar, welcher auch in der natürlichen Sprache eine einzelne Gegebenheit mittels mehrerer Wörter darstellen würde. Deswegen werden Akronyme hier als gesonderte Form der Abkürzung gewertet, und nicht als Sonderform der Konkatenation mehrerer Abkürzungen, welche nur aus einem Buchstaben bestehen. Diese vier oder ähnliche Abkürzungstypen werden auch in mehreren verwandten Arbeiten [NDA+19] [AXX17] [HFB+08] [CDMM12] [JLZZ18] [GGG+12] definiert. Bei einigen ist die Rede von Abkürzungsstrategien [CDMM12] [GGG+12], womit analog der Vorgang des Abkürzens anstelle der Abkürzung selbst beschrieben wird. Einige Arbeiten betrachten Abkürzungen, welche nur aus einem Buchstaben bestehen, als Sonderfall [HFB+08] [AXX17], allerdings wird für solche Abkürzungen stets entweder die gleiche Auflösungsstrategie verwendet wie für Präfixe, oder die gleiche wie für Akronyme. Laut Newman et al. [NDA+19] sind Präfix-Abkürzungen bei weitem die häufigsten in Quelltext (etwa 55,3% aller Abkürzungen), mit großem Abstand zu Wörtern aus denen Buchstaben entfernt wurden (23,9%) und Akronymen (20,8%). Die Zahl der Multiwort-Abkürzungen ist vernachlässigbar, wie Tabelle 5.1 zeigt. Diese Statistiken wurden auf dem Quelltext von fünf Programmen erhoben. Dabei war die höchste Rate an Multiwort-Abkürzungen eines dieser Programme 3%. Die Verteilung der verschiedenen Abkürzungstypen im Quelltext weicht nicht stark von der gesamten Verteilung der Abkürzungen im Quelltext ab. Zumindest die Reihenfolge der Quelltext-Orte, in denen Abkürzungen am häufigsten vorkommen, ist für alle Abkürzungstypen identisch mit den in Abbildung 5.1 beschriebenen Verteilungen aller Abkürzungen, nur die Ausprägung schwankt. So sind, den Beobachtungen von Newman et al. zu Folge, beispielsweise 80,3% aller Präfix-Abkürzungen und 37,6% aller Akronyme im Methodenrumpf zu finden, für beide ist dies aber der größte in irgendeinem Kontext vertretene Anteil.

Beispiel 5.5: Präfix-Abkürzungen

- str *für* string
- int *für* integer

Beispiel 5.6: Abkürzung durch Weglassen von Buchstaben

- cmpr *für* compare
- pntr *für* pointer
- strm *für* stream

Beispiel 5.7: Akronyme

- cn *für* class name
- da *für* dynamic array

Beispiel 5.8: Kombinierte Multiwort-Abkürzungen

- strCmpr *für* string compare
- strstrm *für* string stream

5.2.1 Präfix-Abkürzungen

Für Präfix-Abkürzungen muss zunächst festgestellt werden, wie viele Buchstaben eines Wortes weggeschnitten beziehungsweise behalten werden. Leider scheint es keine Statistik darüber zu geben, auf welche sich hier berufen werden könnte. Was es allerdings gibt, ist eine Statistik von Xu et al. [XXA+18] darüber, wie viel Tippaufwand Abkürzungen im allgemeinen im Schnitt einsparen, also wie viel kürzer als das nicht abgekürzte Wort sie im Schnitt sind. Daraus ergibt sich, dass ca. 70% aller Abkürzungen zwischen 50% und 90% weniger Zeichen enthalten als ihre korrekte Auflösung, woraus sich auf die Wahrscheinlichkeit eines Auflösungskandidaten mit einer gewissen Länge schließen lässt. Einige Arbeiten zum Thema der Abkürzungsauflösung wagen die Vermutung, Abkürzungen, aus welchen Buchstaben weggelassen wurden, würden in erster Linie Vokale weglassen [CDMM12]. Hier sei in Anlehnung daran zunächst die Vermutung aufgestellt, Präfix Akronyme würden häufiger vor Vokalen abgeschnitten, als vor Konsonanten, und ebenso seltener nach Vokalen als nach Konsonanten. Die in Beispiel 5.5 genannten Abkürzungen *str* für *string* und *int* für *integer* würden dieser Vermutung entsprechen. In dem Seitenbeschreibungswerkzeug *a2ps*, welches auch als Probanden-Programm in *LINSEN* [CDMM12] verwendet wird, gibt es zwei weitere Beispiele, welche diesem Prinzip entsprechen. Sowohl *res* für *resolution* in Beispiel 5.9, als auch *tim* für *time* in Beispiel 5.10, enden nach einem Konsonanten und vor einem Vokal, erfüllen also genau wie *str* und *int* beide Vermutungen.

Tabelle 5.2: Anteil aller Präfix-Abkürzungen, welche in bestimmten Quelltext-Bereichen auftreten, laut Newman et al. [NDA+19]

Typen von Deklarationen	Typen von Parametern	Namen von Deklarationen	Namen von Parametern
65.5%	52.5%	75.3%	58.5%

Namen von Ausdrücken	Methodenrumpf	Klassen-Namen und -Felder	Globale Variablen
73.1%	80.3%	67.9%	72.7%

Die Summe ergibt nicht 100%, da einige Abkürzungen mehrfach auftreten.

Beispiel 5.9: Präfix-Abkürzungen aus *a2ps* - *res* für *resolution*

- *res* tritt in einem Parameter der Datei *generate.c* auf.
- Die Auflösung tritt nirgends im Quelltext der selben Datei auf, aber anderswo in Kommentaren des Projektes.

```
if (IS_EMPTY(name) || ustrequ (name, "-"))
{
        file_job->is_stdin = true;
        file_job->name = job->stdin_filename;
        /* Create the buffer in charge of stdin */
        buffer_stream_set (res, stdin, end_of_line);
        /* Ask it to make a sample of the file. */
        tempname_ensure (sample_tmpname);
        buffer_sample_get (res, sample_tmpname);
}
```

Beispiel 5.10: Präfix-Abkürzungen aus *a2ps* - *tim* für *time*

- *tim* tritt in einer lokalen Variable der Datei *generate.c* auf.
- Die Auflösung tritt im Quelltext in der Nähe des Bezeichners auf, in kombinierten Bezeichnern *time_t*, *localtime* und *st_mtime*.
- Die Abkürzung mit weggelassen Buchstaben *tm* tritt in der Nähe auf.

```
time_t tim = statbuf.st_mtime;
tm = localtime (&tim);
memcpy (&(file_job->mod_tm), tm, sizeof (*tm));
```

Statistische Details zum Auftreten von Präfix-Abkürzungen sind in Tabelle 5.2 zu finden. Diese sind im Einklang mit der allgemeinen Verteilung von Abkürzungen in Quelltext, wie sie in Abbildung 5.1 gezeigt werden. Allerdings treten die meisten Abkürzungen im Quelltext mehrfach auf. So tritt in keinem Bereich des betrachteten Quelltextes weniger als die Hälfte aller Präfix-Abkürzungen auf, wobei Typen von Parametern und Namen von Parametern mit 52,5% und 58,5% recht weit abgeschlagen sind, und Methodenrümpfe mit 80,3% deutlich herausstechen.

Tabelle 5.3: Anteil aller Abkürzungen mit weggelassenen Buchstaben, welche in bestimmten Quelltext-Bereichen auftreten, laut Newman et al. [NDA+19]

Typen von Deklarationen	Typen von Parametern	Namen von Deklarationen	Namen von Parametern
51,1%	38,9%	57,8%	45,6%
Namen von Ausdrücken	Methodenrumpf	Klassen-Namen und -Felder	Globale Variablen
55,0%	61,7%	46,7%	52,2%

Die Summe ergibt nicht 100%, da einige Abkürzungen mehrfach auftreten.

5.2.2 Abkürzungen mit weggelassenen Buchstaben

Wie oben schon erwähnt, werden bei Abkürzungen, die entstanden sind, indem aus ihrer Auflösung schlicht Buchstaben weggelassen wurden, zumindest intuitiv, eher Vokale weggelassen als Konsonanten [CDMM12]. Zudem ist es allgemein üblich den ersten Buchstaben eines Wortes immer in eine Abkürzung mit einzubeziehen. Beides könnte damit zusammenhängen, dass Menschen Wörter anhand ihres groben Umrisses erkennen [BB79]. Ein Beispiel aus *a2ps*, wo das Weglassen der Vokale zutrifft, ist *tm* anstelle von *time*, siehe Beispiel 5.11.

Beispiel 5.11: Abkürzung mit weggelassenen Buchstaben aus *a2ps* - *tm* für *time*

- *tm* tritt in einer lokalen Variable der Datei *generate.c* auf.
- Die Auflösung tritt im Quelltext in der Nähe des Bezeichners auf, in kombinierten Bezeichnern *time_t*, *localtime* und *st_mtime*.
- Die Präfix-Abkürzung *tim* tritt in der Nähe auf.

```
time_t tim = statbuf.st_mtime;
tm = localtime (&tim);
memcpy (&(file_job->mod_tm), tm, sizeof (*tm));
```

Zusammenhänge mit dem Weglassen von Vokalen in SMS-Sprache [Wik21c] oder Konsonantenschrift [Wik21a] könnten vermutet werden, sind aber nicht explizit bekannt. Für den umgekehrten Fall, Konsonanten weg zu lassen und Vokale zu behalten, ist es sowohl schwer eine Intuition, als auch Beispiele zu finden. Dies bestärkt zwar die Hypothese der häufiger weggelassenen Vokale, allerdings lässt sich ohne Statistik oder Experimente dazu nichts darüber aussagen, ob diese Intuition einen Nutzen für das Finden richtiger Auflösungen von Abkürzungen bringt. Einige Arbeiten zur Auflösung von Abkürzungen [LBM10] [CDMM12] [MGDP+10] [GGG+12] [HFB+08] nutzen diese Intuition neben anderen Methoden recht erfolgreich, doch geben sie keine Statistik an, wie oft die Intuition der weggelassenen Vokale zutrifft, oder wie oft sie maßgeblich wie eine korrekte Auflösung oder Spaltung des Bezeichners verantwortlich ist. Auch ist in Beispiel 5.11 zu beobachten, dass *tm* die zweite Abkürzung im selben Kontext ist, welche grob das gleiche Konzept, nämlich Zeit, beschreibt. Anstatt bei der Bezeichnung konkret auf die Rolle der jeweiligen Variable einzugehen, wird mal die eine, mal die andere Abkürzung, mit je unterschiedlichen Abkürzungsstrategien verwendet. Das zeigt deutlich, welche Absicht der Entwickler im Bezeichner festzuhalten versuchte, und welche er nicht für nötig erachtete im Bezeichner zu kommunizieren: Sowohl *tim* als auch *tm* beschreiben Zeitpunkte. Diese Information wird im Bezeichner kommuniziert, dass beide diesen Zeitpunkt aber in unterschiedlichen Daten-

Tabelle 5.4: Anteil aller Akronyme, welche in bestimmten Software-Artefakten auftreten, laut Newman et al. [NDA+19]

Kommentare	Projekt	Sprache	Informatik Wörterbuch	Englisch Wörterbuch
21,7%	67,5%	93,6%	21,7%	0,0%

Die Summe ergibt nicht 100%, da einige Akronyme mehrfach auftreten.

Tabelle 5.5: Anteil aller Akronyme, welche in bestimmten Quelltext-Bereichen auftreten, laut Newman et al. [NDA+19]

Typen von Deklarationen	Typen von Parametern	Namen von Deklarationen	Namen von Parametern
29,3%	19,1%	33,8%	21,0%

Namen von Ausdrücken	Methodenrumpf	Klassen-Namen und -Felder	Globale Variablen
26,1%	37,6%	22,3%	23,6%

Die Summe ergibt nicht 100%, da einige Akronyme mehrfach auftreten.

formaten speichern, wobei das eine rechnerisch praktischer und das andere für Menschen zugänglicher ist, hält der Autor entweder für vernachlässigbar, oder für ersichtlich genug. Statistische Details zum Auftreten von Abkürzungen mit weggelassenen Buchstaben, aus den Beobachtungen von Newman et al. [NDA+19], sind in Tabelle 5.3 zu finden. Genau wie bei den Präfix-Abkürzungen in Tabelle 5.2 weicht die Reihenfolge des häufigsten Auftretens in bestimmten Teilen des Quelltextes nicht von der allgemeinen Verteilung von Abkürzungen im Quelltext in Abbildung 5.1 ab. Allerdings fällt auf, dass die Zahlen allgemein nur zwischen 38,9% und 61,7% liegen, das heißt die jeweiligen Kontexte decken einen geringeren Anteil aller Abkürzungen mit weggelassenen Buchstaben ab, als den Anteil der Präfix-Abkürzungen.

5.2.3 Akronyme

Akronyme stellen in weiten Teilen der bisherigen Forschung eine besondere Schwierigkeit dar. In jeder Forschungsarbeit, auf welche in Kapitel 4 Bezug genommen wird, welche ihre Ergebnisse nach Abkürzungstypen klassifiziert, werden für Akronyme die schlechtesten Ergebnisse erzielt. Akronyme, als Kette der Anfangsbuchstaben mehrerer Wörter, bieten den Nachteil, dass für jedes der einzelnen Wörter nur ein einzelner Buchstabe als Indiz dient. Zudem muss ein Akronym, um entsprechend nach einer Auflösung je Buchstabe zu suchen, zuerst als solches erkannt werden. Zwar lassen diese sich oft daran erkennen, dass es, zumindest in natürlicher Sprache, üblich ist sie vollständig groß zu schreiben, doch ist dies gerade in Quelltext, wie Beispiel 5.7 zeigt, nicht immer der Fall, und kann auch auf andere Bezeichner, insbesondere globale Konstanten, zutreffen. Beispiel 5.12 aus *a2ps* zeigt das Akronym *EOF* für *end of file*. Hier fällt auf, dass in einigen Bezeichnern der Begriff ausgeschrieben wird, während er in anderen abgekürzt ist. Zudem ist das Akronym meistens, aber nicht immer, groß geschrieben. Die Verteilung der Orte, in welchen Akronyme am häufigsten im Quelltext auftreten, weist, wie Tabelle 5.5 zeigt, die gleiche Reihenfolge auf, wie die Verteilung aller Abkürzungen, welche in Abbildung 5.1 gezeigt wird. Allerdings beinhalten die einzelnen Quelltextbereiche jeweils einen verhältnismäßig geringen Anteil aller Akronyme, was bedeutet, dass das selbe Akronym seltener an mehreren Quelltextorten auftritt. Was die Suche nach entsprechenden Kandidaten betrifft, ist es, wie bei anderen Abkürzungen auch, sowohl möglich diese im Quelltext zu finden, als auch in anderen Artefakten. Da Akronyme jenseits von Quelltext im allgemeinen Sprach-

gebrauch, im Gegensatz zu anderen Abkürzungstypen, recht beliebt sind, herrscht die Intuition vor, dass Akronyme seltener spontan durch den Entwickler formuliert werden, sondern oftmals allgemein verbreitete, beziehungsweise domänenspezifische, Abkürzungen sind. Dies würde nahe legen, dass das Akronym durch den Entwickler als Fachbegriff für sich wahrgenommen wird, und somit nicht im Quelltext erklärt wird. Somit würde die Auflösung der Abkürzung nicht im Quelltext, sondern in der Dokumentation und anderen Domänen-Artefakten zu finden sein. Gegen die Domäne eines einzelnen Programms als Auflösungsquelle für Akronyme würde sprechen, dass die Akronyme selbst laut Newman et al. [NDA+19] am häufigsten in der Programmiersprache auftreten, wie Tabelle 5.4 zeigt. Somit wäre die Dokumentation der Programmiersprache eine viel versprechende Quelle für die Suche nach der Auflösung.

Beispiel 5.12: Akronym aus *a2ps - EOF für end of file*

- *EOF* tritt in lokalen Variablen und Makros der Datei *lexps.c* auf, meistens vollständig groß geschrieben, aber nicht immer.
- Die Auflösung tritt in Kommentaren weit weg von der Abkürzung auf, teils getrennt durch Bindestriche, oder in kombiniertem Bezeichner *EOB_ACT_END_OF_FILE*.

```
/* Action number for EOF rule of a given start state. */
#define YY_STATE_EOF(state) (YY_END_OF_BUFFER + state + 1)
```

```
#define YY_INPUT(buf,result,max_size) \
if ( YY_CURRENT_BUFFER_LVALUE->yy_is_interactive ) \
{ \
        int c = '*'; \
        size_t n; \
        for ( n = 0; n < max_size && \
        (c = getc( psin )) != EOF && c != '\n'; ++n ) \
        buf[n] = (char) c; \
        if ( c == '\n' ) \
        buf[n++] = (char) c; \
        if ( c == EOF && ferror( psin ) ) \
        YY_FATAL_ERROR( "input in flex scanner failed" ); \
        result = n; \
} \
```

```
*       EOB_ACT_END_OF_FILE - end of file
*       (...)
*/
```

```
case EOB_ACT_END_OF_FILE:
{
        (yy_did_buffer_switch_on_eof) = 0;
        (...)
}
```

5.2.4 Abkürzungen in kombinierten Multiwort-Bezeichnern

Wie die Beispiele anderer Abkürzungstypen zeigen, treten Abkürzungen oftmals als Teil eines Bezeichners, und nicht unbedingt als eigenständiger Bezeichner, auf. Diese Bezeichner stellen also zusätzlich das Problem dar, dass sie zunächst in die einzelnen Bestandteile, ob diese nun abgekürzt sind oder nicht, aufgeteilt werden müssen. Oftmals werden hierbei einzelne Teilbegriffe mit eigener Bedeutung durch Trennzeichen wie Binnenmajus-

kelschreibweise oder Unterstriche getrennt, und oftmals sind einige der Bezeichner keine Abkürzungen sondern nicht abgekürzte ganze Wörter, welche sich durch einfachen Abgleich mit einem Wörterbuch als solche identifizieren lassen. Beides hat den Vorteil, dass es dazu verhilft, die Abkürzungen aus dem restlichen Bezeichner zu isolieren. In Beispiel 5.13 aus *a2ps* wird auf den abgekürzten Bezeichner *fface* eingegangen. *fface* steht für *flag face*, der Bezeichner besteht also aus der Abkürzung *f* für *flag* und dem nicht abgekürzten Wort *face*. Im Bezeichner *sample_tmpname* aus Beispiel 5.14 liegt die Abkürzung *tmp* vor. Dieser Bezeichner ist aus drei Segmenten kombiniert, von denen zwei natürlichsprachliche Wörter sind, und nur eines durch ein Trennzeichen abgegrenzt wird. Dieses Beispiel zeigt also zum einen die Bedeutung von Trennzeichen bei de Spaltung von kombinierten Multiwort-Bezeichnern, und zum anderen, dass diese eben keine Erfolgsgarantie darstellen, sondern durch weitere Spaltmethoden ergänzt werden müssen.

Beispiel 5.13: Kombinierte Multiwort-Abkürzung aus *a2ps* - *fface* für *flag face*

- *fface* tritt in Structs und Instanzen dieser Structs in der Datei *ffaces.c* auf. Außerdem tritt *fface* teils im Plural, so auch im Dateinamen.
- Die Auflösung tritt in Kommentar am Anfang der Datei auf.

```
/*
* ffaces.h
*
* definition of the flagged faces used by \textit{a2ps}
(...)
*/
```

```
/*
* Some predefined ffaces.
*/
struct fface_s String_fface = { String, ff_No_fflag };
struct fface_s Plain_fface = { Plain, ff_No_fflag };
struct fface_s Symbol_fface = { Symbol, ff_No_fflag };
struct fface_s No_fface = { No_face, ff_No_fflag };
```

Beispiel 5.14: Kombinierte Multiwort-Abkürzung aus *a2ps* - *sample_tmpname* für *sample temporary name*

- *sample_tmpname* tritt in einer lokalen Variable der Datei *generate.c* auf.
- Die Auflösung für *tmp* tritt nicht in selber Datei auf, nur anderswo in Kommentaren im Projekt

```
/* Name of the file in \textit{which} the tmp sample file is stored
   . */
char *sample_tmpname = NULL;
```

```
/* Retrieve file modification date and hour */
if (IS_EMPTY(name) || ustrequ (name, "-"))
{
        file_job->is_stdin = true;
        file_job->name = job->stdin_filename;
        /* Create the buffer in charge of stdin */
        buffer_stream_set (res, stdin, end_of_line);
        /* Ask it to make a sample of the file. */
        tempname_ensure (sample_tmpname);
        buffer_sample_get (res, sample_tmpname);
}
```

5.2.5 Abkürzungserkennungsmethoden aus verwandten Arbeiten

Um abgekürzte Bezeichner als solche zu erkennen, kann es helfen, zunächst zu bestimmen, was diese Bezeichner nicht sind, nämlich Wörterbuch-Wörter oder von der Quelltext-Sprache vorgegebene Bezeichner. So wird dies auch in der Arbeit von Lawrie et al. [LFB07] umgesetzt, wo zunächst über Wahrscheinlichkeitsverteilungen festgestellt wird, ob ein Bezeichner üblich oder unüblich ist. Wird so festgestellt, dass ein Bezeichner unbekannt ist, gibt es bislang kaum Methoden um zu bestimmen, ob er Abkürzungen enthält, ohne zu wissen, welche Wörter abgekürzt worden sind. Einige Verfahren gehen davon aus, dass es sich bei einem Wort um eine Abkürzung handelt, wenn dieses nicht in einem Wörterbuch zu finden ist. In *Normalize* [LB11], *LINSEN* [CDMM12] und der Arbeit von Jiang et al. [JLZZ18] wird dies für die weichen Wörter, während oder nach Spaltung der Bezeichner durchgeführt. In *TRIS* [GGG+12] wird das Nichtabkürzen eines Wortes als eine Abkürzungsstrategie betrachtet, welche zum Zeitpunkt der Auflösung zurück verfolgt werden kann. In einem Verfahren von Zhang et al. [ZSST11], welches sich ausschließlich mit Akronymen befasst, wird davon ausgegangen, dass diese vollständig groß geschrieben werden. Liu et al. [LGM+15] nutzen reguläre Ausdrücke um die Abkürzungen aus einem Text zu extrahieren.

Die Erkennung von Akronymen anhand dessen, dass sie vollständig groß geschrieben werden, ist erfahrungsgemäß nicht zuverlässig genug. Sowohl weil andere Bezeichner, konventionsgemäß vor allem Konstanten, auch vollständig groß geschrieben werden, als auch weil Akronyme mitunter klein geschrieben werden. Die Erkennung von potentiellen Abkürzungen durch die Erkennung von Wörtern, welche nicht in einem Wörterbuch vorkommen, ist einfach umzusetzen und bietet offensichtliche Vorteile für Genauigkeit und Performanz. Dies gilt sowohl für den Abgleich ganzer Bezeichner, als auch für den Abgleich von darin befindlichen Token, welche durch Spaltung an Trennzeichen gewonnen wurden. Beim Abgleich anderer, wenn nicht aller möglichen Spaltungen des Bezeichners, bietet dies außerdem den Vorteil, dass die Spaltung hierdurch schon entschieden, oder teilweise entschieden werden kann. Im Fall, dass eine Abkürzung oder ein Teilbezeichner zufällig homonym

mit einem realen Wort ist, könnte das Nichtauflösen von Wörterbuch-Wörtern allerdings der Ausbeute des Verfahrens schaden. Darüber hinaus komplexere Methoden zur Erkennung von Abkürzungen zu entwickeln erscheint überflüssig, da der Aufwand die implizite Erkennung beim Versuch der Auflösung übersteigen könnte, ohne bessere Ergebnisse zu liefern.

5.2.6 Weitere Methoden zur Erkennung von Abkürzungen in Quelltext

In Abschnitt 5.1 und Abschnitt 5.2 wird darauf eingegangen, dass Abkürzungen mitunter gewisse Formen aufweisen, an welchen sie wiedererkannt werden könnten. Die verschiedenen Abkürzungstypen, welche in Abschnitt 5.2 vorgestellt wurden, könnten eventuell anhand ihrer spezifischen Eigenschaften erkannt werden. Zudem gibt es gewisse Intuitionen, welche scheinbar für die verschiedenen Abkürzungstypen gelten. Beispielsweise könnten Abkürzungen mit weggelassenen Buchstaben dadurch erkannt werden, dass sie verhältnismäßig viele Konsonanten pro Vokal enthalten. Präfix-Abkürzungen könnten daran erkannt werden, dass sie mit Konsonanten enden, beziehungsweise mit solchen Konsonanten enden, mit denen andere Wörter seltener enden. Allerdings würde dies voraussetzen, dass die entsprechenden Vermutungen tatsächlich zutreffen, was nicht belegt ist. Zudem kann es sein, dass diese Vermutungen zwar oft, aber nicht immer gelten, und somit einige Abkürzungen unerkannt blieben. Und zuletzt ist es gut möglich, dass der Aufwand den Gewinn nicht wert wäre, insbesondere im Verhältnis zur impliziten Erkennung von Abkürzungen nach dem Versuch der Auflösung.

5.3 Spaltung

Wie oben Abschnitt 5.2.4 bereits behandelt, können Abkürzungen auch in Bezeichnern auftreten, welche aus mehreren Token zusammengesetzt sind. Hier muss also festgestellt werden, ob ein Bezeichner aus mehreren Wörtern zusammengesetzt ist, wo die Grenzen zwischen diesen Wörtern liegen, bei welchen es sich um Abkürzungen handelt und welchen Typ diese jeweils haben. Für zusammengesetzte Bezeichner gibt es einige Standards. Java-Entwickler werden beispielsweise dazu angehalten, die meisten Multiwort-Bezeichner mit Binnenmajuskel zu trennen, und Konstanten vollständig groß zu schreiben und mit Unterstrichen zu trennen [Ora21]. Google empfiehlt in C++ Typdeklarationen und Methodennamen mit Binnenmajuskelschreibweise und großen Anfangsbuchstaben zu schreiben, Konstanten genauso aber mit einem vorangestellten kleinen *k*, und Variablennamen mit Trennung durch Unterstrich [Goo21a]. Für Python wird empfohlen Klassennamen in Binnenmajuskelschreibweise mit großen Anfangsbuchstaben zu schreiben, während Variablennamen und Methodennamen vollständig klein und gegebenenfalls mit Trennung durch Unterstrich geschrieben werden sollten [RWC21]. Leider halten Entwickler sich nicht immer an derartige Vorgaben, weshalb sich nicht auf die Spaltung an Trennzeichen wie Binnenmajuskelschreibweise oder Unterstrichen verlassen werden kann. In einzelnen Fällen sind auch Zahlen Teil eines zusammengesetzten Bezeichners, sodass diese als Trennzeichen verwendet werden können, allerdings ist dies auch nicht die Regel. Ein weiteres Indiz auf Stellen im Bezeichner, an welchen Spaltungen notwendig sind, sind natürlichsprachliche Token im Bezeichner, welche durch einfachen Abgleich mit einem Wörterbuch erkannt werden können. Im Bezeichner *sample_tmpname* in Beispiel 5.14 liegt die Abkürzung *tmp* zwischen einem Unterstrich und dem natürlichen Wort *name*. Trotz des fehlenden Trennzeichens nach rechts hin sollte die Isolierung der Abkürzung also kein großes Problem darstellen, da das natürlichsprachliche Token isoliert werden kann. Doch ist nicht davon auszugehen, dass zwischen zwei Abkürzungen immer ein solches nicht abgekürztes Token zu finden ist. Für die Spaltung eines Bezeichners in seine Teilbezeichner reicht die Spaltung an offensichtlichen Trennzeichen also nicht aus. Wenn es keine nicht abgekürzten Token im Bezeichner

gibt, welche erkannt werden und so Stellen zur Spaltung gefunden werden könnten, ist dies vielleicht mit den enthaltenen Abkürzungen möglich. Die in Abschnitt 5.2 bereits behandelte Abkürzungserkennung wird hier also erneut zum Thema. Allerdings wird diesbezüglich in Abschnitt 5.2.5 und Abschnitt 5.2.6 bereits festgestellt, dass das explizite Erkennen von Abkürzungen, zumindest außerhalb der Frage der Spaltung, angesichts der impliziten Erkennung durch den Versuch der Auflösung, den Aufwand vermutlich nicht wert ist, und dass es wenig Anhaltspunkte gibt um dies zu versuchen. Es bleibt analog also die Möglichkeit, die Spaltung zusammen mit der Auflösung zu entscheiden. Das heißt also, dass verschiedene, wenn nicht alle möglichen Spaltungen in Erwägung gezogen werden, anhand welcher Auflösungskandidaten generiert werden, und mit der Auswahl des besten Kandidaten implizit die Spaltung entschieden wird. Dies könnte Nachteile für die Laufzeit haben, aber ist für das Ergebnis ebenso vielversprechend wie das Auflösungsverfahren an sich, von welchem ohnehin die Güte des Gesamtverfahrens abhängt.

5.3.1 Spaltungsmethoden aus verwandten Arbeiten

Viele Verfahren spalten zunächst die Bezeichner entlang bestimmter Trennzeichen in harte Wörter, und mithilfe verschiedener anderer Beobachtungen in weiche Wörter, um dann die einzelnen weichen Wörter mehr oder weniger unabhängig voneinander aufzulösen. Einige Arbeiten verwenden gierige Algorithmen zur Spaltung [LFB07] [FBL06], doch neigen diese dazu, zu viel zu spalten [FBL06] [EHPVS09] [LBM10]. Feild et al. [FBL06] vergleichen für die Spaltung einen gierigen Algorithmus mit einem neuronalen Netz. Der gierige Algorithmus erzielte in einem Versuch, auf 4.000 Bezeichnern die zufällig aus 186 Programmen ausgewählt wurden, eine Genauigkeit von 75,69%, und war damit schlechter als das neuronale Netz mit 78,12% Genauigkeit. Dafür benötigt letzteres aber auch manuell gekennzeichnete Trainingsdaten, und ist von diesen Trainingsdaten abhängig und damit anfälliger für geänderte Kontexte. Im Verfahren *Samurai* von Enslen et al. [EHPVS09] werden Bezeichner zunächst anhand von Trennzeichen, wie Binnenmajuskelschreibweise oder Unterstrichen, in harte Wörter aufgeteilt, und diese dann anhand einer Bewertungsfunktion, welche auf den Frequenzen der Auflösungskandidaten der weichen Wörter im Quelltext, beziehungsweise einer Sammlung von Quelltexten, basiert, wiederum in weiche Wörter unterteilt. Damit haben Enslen et al. in einem Experiment mit 8.466 Bezeichnern, welche zufällig aus 9.000 Programmen gewonnen wurden, eine Genauigkeit von 97% erzielt. Lawrie et al. haben ein eigenes Spalt-Verfahren namens *GenTest* [LBM10] entwickelt, welches nach Spaltung an Trennzeichen in harte Wörter diese in weiche Wörter aufspaltet, und zuerst eine Liste aller möglichen Spaltungen generiert, und diese dann bewertet. Für die Bewertung werden gewisse statistische Charakteristika der weichen Wörter, wie Anzahl weicher Wörter im harten Wort, durchschnittliche Länge oder größte Länge eines weichen Wortes, externe Informationen, wie statistische Informationen über die weichen Wörter im Wörterbuch, und interne Informationen, wie statistische Informationen über die weichen Wörter im Quelltext, verwendet. Laut den Ergebnissen der Evaluation erreichen sie damit, auf 4.000 Bezeichnern die zufällig aus 186 Programmen ausgewählt wurden, eine Genauigkeit von 81,82%, was besser ist als das Ergebnis von *Samurai* auf dem gleichen Datensatz mit 70,32%. Hier ist allerdings zu beachten, dass der Datensatz, auf welchem *GenTest* und *Samurai* hier verglichen werden, weit kleiner ist als der, auf welchem *Samurai* mit einem weit besseren Ergebnis evaluiert wurde. Allerdings wurde *GenTest* nicht auf jenem Datensatz evaluiert, das bedeutet also nicht, dass *GenTest* deswegen schlechter ist.

In *LINSEN* [CDMM12] wird die Spaltung zunächst anhand natürlicher Wörter im Bezeichner vollzogen. Wo die übrigen Teile des Bezeichners zu spalten sind, wird basierend auf der Annahme entschieden, dass auch die Abkürzungen, welche eine Multiwort-Abkürzung zusammensetzen, anderswo im Quelltext alleinstehend auftreten. Die Multiwort-Bezeichner werden also dort gespalten, wo andere Bezeichner aus dem Quelltext in ihnen auftreten.

LINSEN hat bei der Spaltung von Bezeichnern und der Auflösung von Abkürzungen aus allen betrachteten Beispielquelltexten bessere Ergebnisse erzielt, als die Verfahren von Madani et al. und Lawrie et al. Bei der Evaluation auf 487 Bezeichnern, welche aus dem Quelltext des Programmes *which* extrahiert wurden, wurde mit 64,6% eine höhere Genauigkeit erzielt als durch *GenTest*, welches auf dem gleichen Datensatz eine Genauigkeit von 58% erzielt. Andere Verfahren entscheiden die Spaltung, oder auch den Abkürzungstyp, erst anhand der Frage, für welche Bezeichner oder Token sie Auflösungen finden [GGG+12] [HFB+08]. Bezüglich des Ergebnisses der Evaluation von *LINSEN*, laut welchem *LINSEN* genauer ist als *GenTest*, sollte allerdings beachtet werden, dass der Datensatz, auf welchem diese Evaluation erfolgt ist, um einiges kleiner ist als der, in welchem *GenTest* weit besser abschneidet. Allerdings ist die Güte von *LINSEN* auf jenem größeren Datensatz nicht bekannt, weshalb das nicht bedeutet, *GenTest* sei unbedingt besser.

Das Verfahren *TRIS* [GGG+12] von Guerrouj et al. soll, mithilfe einer gerichteten Baumstruktur, Bezeichner sowohl spalten, als auch expandieren. Evaluiert wurde aber nur dessen Fähigkeit zur Spaltung. Die Idee hinter dem Verfahren ist, einen Baum aller natürlichsprachlichen Wörter im Quelltext, und derer in einer Ontologie, welche für repräsentativ für die Domäne gehalten wird, aufzubauen. Dieser Baum wird um alle Abkürzungen erweitert, die, basierend auf den natürlichsprachlichen Wörtern und einer Auswahl von Abkürzungsstrategien, denkbar wären. Abkürzungen, beziehungsweise deren mögliche Spaltungen, werden in dem Baum gesucht. Die Spaltung wird hier anhand dessen entschieden, auf welche Abkürzungen aus diesem Baum sich mögliche Token des Bezeichners abbilden lassen. Auf dem von Lawrie et al. bereitgestellten Datensatz [LBM10] erreicht *TRIS* damit eine Genauigkeit von 86%, womit es sowohl *GenTest* als auch *Samurai*, welche 82% und 70% Genauigkeit erreichen, überbietet. Dass *TRIS* nur auf seine Fähigkeit zur Spaltung hin evaluiert wurde, hat vermutlich den Hintergrund, dass die Spaltung bei *TRIS* aus der gewählten Auflösung resultiert, und sich somit aus der Spaltung eine Einschätzung der Qualität der Auflösung ableiten lässt. Allerdings ist dies für den Vergleich mit anderen Auflösungsverfahren eher unglücklich.

Allgemein ist es zwar gut, dass die verschiedenen Verfahren oftmals gegeneinander direkt verglichen wurden, allerdings ist es leider recht selten der Fall, dass sie auf den gleichen Datensätzen evaluiert wurden, auf denen sie jeweils die besten Ergebnisse erzielt haben. Nicht nur sind die Datensätze unterschiedlich, sie bewegen sich außerdem in völlig unterschiedlichen Größenordnungen, was den Vergleich noch weniger zuverlässig macht. Bezüglich *TRIS* ist der Vergleich etwas aussagekräftiger, weil der gleiche Datensatz für die Evaluation genutzt wurde, wie in der Arbeit zu *GenTest* und deren Vergleich zu *Samurai*.

Sowohl am Beispiel von *TRIS* [GGG+12], als auch später in Abschnitt 5.4.3 wird deutlich, dass die Methode, die Spaltung zusammen mit der Auflösung zu entscheiden, bislang am besten funktioniert. Die Verfahren von Alatawi et al. [AXX17] [AXY18] liefern, laut ihrer eigenen Evaluation, die besten Auflösungsergebnisse ohne eine explizite Spaltung im Voraus, während *TRIS* explizit im Bezug auf die Güte der Spaltungen evaluiert wurde, welche ebenfalls über die Güte der Auflösungskandidaten im Baum entschieden wird, und die besten Spaltergebnisse der betrachteten Verfahren vorweist.

5.4 Auflösung

In Abschnitt 5.1 werden drei Schritte zur Auflösung von Abkürzungen vorgeschlagen: die Erkennung, die Spaltung und zuletzt die Auflösung. Es wird aber sowohl bezüglich der Erkennung in Abschnitt 5.2.6, als auch bezüglich der Spaltung in Abschnitt 5.3.1, darauf eingegangen, dass diese wiederum anhand der Auflösung entschieden werden können, die beste Herangehensweise sein könnte. Dennoch kann, auch bei Entscheidung der Erkennung und Spaltung erst nach der Auflösung, für die Auflösung angenommen werden,

Tabelle 5.6: Auftreten der richtigen Auflösung einer Abkürzung, in verschiedenen Artefakten, für insgesamt 3.067 Bezeichner aus fünf Programmen, laut Newman et al. [NDA+19].

Sprach-Dokumentation	Projekt-Dokumentation	Englisch-Wörterbuch	Methoden-Rümpfe und Parameter
23,7%	19,6%	17,1%	15,6%

Kommentare	Informatik-Wörterbuch	Methoden-namen	Globale Bezeichner	Klassennamen und -Felder
12,7%	9,5%	1,1%	0,4%	0,4%

dass Kandidaten, für die Spaltung eines Bezeichners und für potentiell darin enthaltene Abkürzungen, gegeben sind. Es muss allerdings der Fall berücksichtigt werden, dass beim Versuch der Auflösung der entsprechende Abkürzungs-Kandidat verworfen wird, also dass im Auflösungsschritt entschieden wird, dass ein Bezeichner oder ein Teil davon nicht als Abkürzung angesehen wird. Die Auflösung einer gegebenen Abkürzung kann wiederum in zwei Schritte unterteilt werden: das Finden von Kandidaten zur Auflösung, und die Entscheidung, welches der beste Kandidat ist.

5.4.1 Finden von Kandidaten

Die Suche nach den Kandidaten benötigt einerseits Wissensquellen, in welchen die Kandidaten zu finden sind, und andererseits Methoden, um sie auch als Kandidaten zu identifizieren. Hierbei gilt es einerseits, genug Kandidaten zu berücksichtigen, damit der richtige mit hoher Wahrscheinlichkeit dabei ist, aber andererseits eine möglichst enge Vorauswahl an plausiblen Kandidaten zu treffen, damit der richtige mit größerer Wahrscheinlichkeit ausgewählt wird. Was das im Einzelfall bedeutet, hängt von den Methoden zur Identifikation von Kandidaten, und von der Entscheidung der Auflösung ab. Diese Vorauswahl wird bereits mit der Auswahl der betrachteten Wissensquellen reguliert.

5.4.1.1 Wissensquellen

Eine mögliche Wissensquelle für die suche nach Auflösungskandidaten ist der Quelltext selbst. Sollen möglichst viele Kandidaten gefunden werden, beispielsweise auf Grund von großem Vertrauen in die Funktion zur Auswahl des richtigen Kandidaten, bietet es sich an einen möglichst großen Kontext, beziehungsweise den gesamten Quelltext, nach Kandidaten zur Auflösung einer Abkürzung zu durchsuchen. Soll bei der Suche nach Kandidaten bereits die Auswahl eingegrenzt werden, bietet sich ein möglichst konkreter Kontext an, von dem erhofft wird, dass er möglichst die richtige Auflösung und keine oder nur wenige weitere Kandidaten enthält. Einerseits kann in bestimmten Bereichen des Quelltextes gesucht werden, von denen besonders gute Ergebnisse erhofft werden. Eine Studie von Newman et al. [NDA+19] legt, wie Abbildung 5.1 zeigt, beispielsweise nahe, dass Abkürzungsauflösung in Quelltextbezeichnern allgemein am häufigsten in Methodenrümpfen auftreten. Es könnte auch, auf die Vermutung hin, dass die Auflösung in der Nähe der Abkürzung vorkommt oder sogar bewusst erklärt wird, in unmittelbarer Nähe der jeweiligen Abkürzung gesucht werden. Diese Nähe zur Abkürzung lässt sich auf verschiedene Weise definieren. Dies könnte über den Abstand in Zeichen oder Zeilen definiert werden, aber auch über den Quelltextblock, oder gar über Zuweisungen und Methoden-Aufrufe. Bei der Abkürzung *res* aus Beispiel 5.9 ist die Auflösung *resolution* nur in den Kommentaren anderer Dateien zu finden, und zwei mal in Fließtext in Kommentaren, welcher eine Funktionsweise eines Features erklärt. Außerdem tritt die Auflösung einmal in der Beschreibung des Bezeichners *$dlsyms* auf, obwohl dieser keine Abkürzung enthält, welche konkret zu *resolution* aufzulösen wäre. In Beispiel 5.12 ist das Akronym *EOF* eine Abkürzung für *end*

of file. In einigen Bezeichnern in der Nähe wird der Begriff ausgeschrieben, während er in anderen abgekürzt ist. Das Akronym wird außerdem nie explizit in Kommentaren erklärt, dennoch taucht die Auflösung in Kommentaren auf, welche andere Bezeichner oder andere Zusammenhänge erklären. Der abgekürzte Bezeichner *fface* in Beispiel 5.13 steht für *flag face*, der Bezeichner besteht also aus der Abkürzung *f* für *flag* und dem nicht abgekürzten Wort *face*. Im plural *ffaces* stellt er auch den Namen einer Quelltext-Datei dar. Dort wird er gleich im ersten Kommentar der Datei erklärt.

Dass ein Bezeichner, welcher nicht abgekürzt vorliegt, und einer, welcher das Gleiche abgekürzt beschreibt, aufeinander verweisen, kann beispielsweise daher rühren, dass Bezeichner eindeutig sein müssen, was ein Grund dafür sein kann, dass überhaupt einer der Bezeichner abgekürzt wurde. Dies würde sowohl für die Suche im entsprechenden Quelltextblock, also auch in der Nähe, gemessen in Bezeichnern oder Zeilen, sprechen. Bei *tim* aus Beispiel 5.10 ist die Auflösung direkt in Bezeichnern in der Nähe zu finden, allerdings nur in kombinierten Multiwort-Bezeichnern. Auch spart das Weglassen eines einzelnen Buchstabens so wenig Tippaufwand, dass es vermuten lässt, der Bezeichner sei nur aus Versehen abgekürzt worden, oder um der Notwendigkeit der eindeutigen Benennung nachzukommen. In unmittelbarer Nähe dazu tritt außerdem die Abkürzung *tm* aus Beispiel 5.11 auf, welche das gleiche beschreibt. Bei den meisten eben genannten Abkürzungen, welche alle aus *a2ps* stammen, fällt auf, dass die Entwickler keinen gezielten Hinweis auf die Auflösung in den Kommentaren für nötig erachten.

Zudem hat jeder Quelltext-Kontext eine gewisse zusammenhängende Semantik, für welche es, ebenso wie für die Domäne, naheliegend ist, dass sie Begriffe, die hier von Bedeutung sind, enthält, was eben auch die Auflösungen der darin befindlichen Abkürzungen betrifft. Dies würde ebenfalls für die Suche innerhalb eines Quelltext-Blockes sprechen, aber auch für das Verfolgen von Methoden-Aufrufen und anderen Verweisen. Für den Bezeichner *sample_tmpname* in Beispiel 5.14 ist die mutmaßlich richtige Auflösung *temporary*, für das wiche Wort „tmp“, nirgends in der selben Quelltextdatei zu finden, nur in anderen Dateien und Kontexten im Projekt. Es könnte hier allerdings argumentiert werden, dass es sich bei „tmp“ um eine wohl bekannte Abkürzung handelt - wenn denn das Auflösungswerkzeug eine entsprechende Liste einsetzt.

Aber nicht nur im Quelltext, auch in anderen, das Programm oder die Domäne betreffenden Artefakten, können Auflösungskandidaten gefunden werden. Entwickler schreiben Programme, welche in gewissen Domänen genutzt werden sollen. Zudem verfügen sie über sprachliches Wissen, welches sie nicht zwingend unmittelbar im Programm explizit angeben, oder welches sie, bewusst oder unbewusst, bei jedem Leser voraussetzen. Der Gedankengang des Entwicklers, in seiner Formulierung von Quelltext-Bezeichnern, bezieht also eventuell Sprachinformationen mit ein, welche in anderen Dokumenten als dem Quelltext selbst zu finden sind. Unter Umständen sind diese gar nicht im Quelltext zu finden, oder überhaupt nur in einem bestimmten Dokument. Um nach Auflösungskandidaten zu suchen, ergibt es also durchaus Sinn weitere Wissensquellen zu berücksichtigen. Diese können die Programm-Dokumentation, die Dokumentation der Programmiersprache, domänenspezifische Literatur oder allgemeine Wörterbücher sein. Newman et al. [NDA+19] haben hierzu eine Studie veröffentlicht, welche das Auftreten der richtigen Auflösungen von Abkürzungen in verschiedenen Artefakten diskutiert, wie in Tabelle 5.6 zu sehen ist.

Daraus lässt sich beim Einsatz mehrerer Wissensquellen eine angemessene Priorisierung ableiten. Es könnte außerdem ein Hinweis darauf sein, welche Sachverhalte in Quelltextbezeichnern kommuniziert werden. Wenn eine Auflösung in der Dokumentation der Programmiersprache zu finden ist, spricht das vermutlich dafür, dass eher technische Aspekte der Software kommuniziert werden, als Details der Domäne in welcher die Software eingesetzt wird. Die Projektdokumentation oder Domänenliteratur enthalten die Beschreibung des

Tabelle 5.7: Anteil richtiger Abkürzungsauflösungen, welche exklusiv in einem bestimmten Artefakt auftreten, für insgesamt 69 Bezeichner aus fünf Programmen, laut Newman et al. [NDA+19].

Sprach-Dokumentation	Projekt-Dokumentation	Englisch-Wörterbuch	Methoden-Rümpfe und Parameter
69,6%	15,9%	1,4%	0,0%

Kommentare	Informatik-Wörterbuch	Methoden-namen	Globale Bezeichner	Klassennamen und -Felder
2,9%	1,4%	8,7%	0,0%	0,0%

Zwecks der Software, die darin enthaltenen Wörter im Quelltext wiederzufinden spricht also dafür, dass die entsprechenden Bezeichner das für den Nutzer sichtbare Verhalten der Software widerspiegeln. Zudem lässt sich aus diesen Statistiken schließen, wie dringend welches Artefakt in die Auflösungsstrategie mit einbezogen werden sollte, um die richtigen Auflösungen zu finden. Die Autoren behandeln hier auch das exklusive Auftreten einiger Auflösungen in einigen Artefakten, wie Tabelle 5.7 zeigt. Diese Beobachtung ist besonders wertvoll, da die entsprechenden Abkürzungen mit keinem anderen der betrachteten Artefakte hätten aufgelöst werden können. Andererseits beträgt der Anteil aller exklusiv in einem einzigen Artefakt auftretenden Auflösungen nur 69 von 3.067 Auflösungen, also nur etwa 2%. Doch könnte diese Fragestellung auch umgekehrt formuliert werden: Welche Auflösung tritt in gewissen Artefakten nicht auf? Denn wird bei der Auflösung von Abkürzungen nur eine oder nur sehr wenige Wissensquellen benutzt, kann auch bei nicht exklusiv in einer einzigen Wissensquelle auftretenden Auflösungen das Problem entstehen, dass keine der Wissensquellen, in welchen diese Auflösung auftritt, berücksichtigt wurde. Gibt es also Gründe dafür, nur eine begrenzte Anzahl an Wissensquellen zu nutzen, sprechen die Ergebnisse von Newman et al. also dafür, sich auf die Sprachdokumentation und vielleicht die Projektdokumentation zu beschränken. Gründe für eine solche Einschränkung können einerseits die Vermeidung falscher Kandidaten, andererseits die Verkürzung der Rechenzeit sein. Noch konkreter könnte es interessant sein, welche Wissensquelle sich für welche Abkürzung besonders eignet. Der Abkürzungstyp, oder die Stelle im Quelltext an welcher eine Abkürzung zu finden ist, könnten Rückschlüsse über die Wissensquelle erlauben, in welcher eine Auflösung zu finden sein könnte. In Abschnitt 5.2 wird auf die Vermutung eingegangen, dass Akronyme öfter allgemein verbreitete Abkürzungen sind, als dass sie spontan durch den Entwickler definiert werden. Dies ließe vermuten, dass die Auflösung eines Akronyms eher in Domänen-Literatur oder der Dokumentation zu finden ist, als im Quelltext. Darüber, ob es Zusammenhänge zwischen dem Ort einer Abkürzung und einer angemessenen Wissensquelle gibt, ist nicht viel bekannt. In einer Arbeit von Jiang et al. [JLZZ18] wird beobachtet, dass Auflösungen zu Abkürzungen in abstrakten Parametern oftmals in den entsprechenden tatsächlichen Parametern oder dem Datentyp des abstrakten Parameters zu finden seien. Weitere ähnliche Zusammenhänge könnten existieren, auf denen aufbauend entsprechend konkrete Auflösungsverfahren entwickeln werden könnten. Allerdings befinden sich laut der Studie von Newman et al. [NDA+19] die meisten Abkürzungsauflösungen, welche innerhalb des Quelltextes zu finden sind, ebenso wie die Abkürzungen selbst, in den Methodenrümpfen. Die Suche innerhalb des selben Methodenrumpfes, in der auch die Abkürzung gefunden wurde, bietet also großes Potential den richtigen Kandidaten zu enthalten. Ist dieser innerhalb der Auswahl der gefundenen Kandidaten, muss dieser für die richtige Auflösung nur noch unter den anderen Kandidaten identifiziert werden. Mehr solcher Zusammenhänge für die Kandidatensuche zu identifizieren erscheint daher redundant.

5.4.1.2 Suche nach Kandidaten in Wissensquellen

Ist die Information gegeben, aus welcher Wissensquelle die Auflösung einer Abkürzungen gewonnen werden soll, bleibt die Frage der konkreten Abbildung einer Abkürzung auf Auflösungskandidaten. Um einen natürlichsprachlichen Term als Auflösungskandidaten in Erwägung zu ziehen, muss abgewogen werden, ob dies plausibel ist. Es wird in Abschnitt 5.1 bereits darauf hingedeutet, dass Entwickler ihre Quelltext-Bezeichner nicht willkürlich abkürzen. Die Abkürzungen basieren auf ihrer natürlichsprachlichen Auflösung. Somit ist es naheliegend, dass eine gewisse Ähnlichkeit im Hinblick auf auftretende Buchstaben und deren Reihenfolge erkennbar ist. Hinzu kommt, dass nicht nur die Begriffe an sich, sondern auch die Art und Weise wie ein Entwickler diese abkürzt, unweigerlich von seiner Prägung abhängt, welche er mit anderen Menschen teilt, insbesondere mit anderen Menschen, welche in der gleichen Programmiersprache programmieren, die gleiche natürliche Sprache sprechen oder wenigstens im Quelltext verwenden, oder in der gleichen Domäne arbeiten. Somit können der Domäne, der Programmiersprache, dem Quelltext und weiteren Wissensquellen nicht nur Auflösungskandidaten in Form natürlichsprachlicher Begriffe und bekannte Abkürzungen entnommen werden, sondern auch verallgemeinerbare oder zumindest wiederkehrende Muster bezüglich der Verwendung, der Form und des Auftretens von Abkürzungen. Der erste Buchstabe eines natürlichen Begriffs verbleibt beispielsweise fast immer in der Abkürzung. Dass in der Abkürzung Buchstaben vorkommen, die in ihrer Auflösung nicht vorkommen, ist allgemein abwegig, es sei denn es dient der Unterscheidung zweier ähnlicher Bezeichner. Doch auch hier ist eher mit der Verwendung von Sonderzeichen oder Zahlen zu rechnen, oder einer noch kürzeren Abkürzung, als mit dem Hinzufügen von Buchstaben. Und zuletzt ist damit zu rechnen, dass die Buchstaben in einer Abkürzung in der gleichen Reihenfolge vorkommen, wie in ihrer Auflösung. Je nach Bedarf an Ausbeute oder Präzision beim Finden plausibler Kandidaten lassen sich diese Merkmale also schon bei der Kandidatensuche, oder erst bei der Auswahl des besten Kandidaten berücksichtigen.

5.4.2 Auswahl des besten Kandidaten

Sind nun einige Auflösungskandidaten für eine Abkürzung gefunden worden, bleibt das Problem der Entscheidung des besten Kandidaten, beziehungsweise der richtigen Auflösung. Als Beispiel soll hier der abgekürzte Bezeichner *fface* aus Beispiel 5.13, welcher für *flag face* steht, dienen. *fface* ist selbst bereits ein kombinierter Multiwort-Bezeichner, tritt aber wiederum in anderen Bezeichnern wie *fface_e* und *fface_s* auf. Hier sind *e* und *s*, durch die Trennung mit Unterstrich, offensichtlich als eigene abgekürzte Terme zu verstehen, was gerade im Fall von *s* die Frage aufwirft, wie ein Auflösungsverfahren erkennen soll, ob es *ffaces* lieber als Plural von *fface*, oder als andere Schreibweise für *fface_s* interpretieren soll. Solche Situationen zeigen, dass bei derartigem erneuten Auftreten einer Abkürzung nicht davon ausgegangen werden sollte, dass sie das gleiche meint wie anderswo. Denn offenbar meint nicht jedes *s* im Quelltext den als *s* abgekürzten Term in *fface_s*. Ganz in der Nähe im Quelltext tritt der Bezeichner *ff_No_fflag* auf. Wozu sollte *fflag* nun korrekterweise aufgelöst werden? Eine Nutzung der anderen Abkürzungen im Kontext könnte, je nach Umsetzung, zu *face-flag* oder *flag-flag* führen. Noch unklarer ist es bei *ff*, was nun *flag-face*, *face-flag*, *flag-flag* oder *face-face* meinen könnte, oder doch etwas völlig anderes.

Um den besten Kandidaten aus einer Menge auszuwählen, müssen diese miteinander verglichen werden. Es ist also hilfreich, die Güte der jeweiligen Kandidaten quantifizieren zu können. Die Frage nach der Wissensquelle, beziehungsweise dem Kontext im Quelltext, welche in Abschnitt 5.4.1.1 für das Finden von Kandidaten vorgeschlagen wird, kann auch als Gütemaß, oder als ein Faktor für die Güte betrachtet werden. Beispielsweise könnte ein Kandidat aus der Sprachdokumentation einem Kandidaten vorgezogen werden, welcher nur in einem allgemeinen Wörterbuch zu finden ist, es sei denn, dass andere Qualitätsmaße

diese Priorisierung wieder verschieben. In Abschnitt 5.2 wird außerdem auf verschiedene Abkürzungstypen eingegangen, welche unterschiedlich oft im Quelltext vorkommen. Die Frage nach dem Abkürzungstyp, den ein gewisser Auflösungskandidat voraussetzt, ist also wiederum ein legitimer Faktor für die Güte des Kandidaten. Würde ein Auflösungskandidat also bedeuten, dass es sich bei der Abkürzung um eine Präfix-Abkürzung handelt, während ein anderer Auflösungskandidat eine Abkürzung mit weggelassenen Buchstaben voraussetzt, würde dies für ersteren Kandidaten als die richtige Auflösung sprechen. Auch sonst können verschiedene Statistiken, sowohl bezüglich der Abkürzung, der Auflösung, der Quelle, oder bezügliche des Zusammenhangs, in dem diese zueinander stehen, Rückschlüsse über die Wahrscheinlichkeit ermöglichen, dass ein Kandidat die richtige Auflösung für die Abkürzung darstellt. Wie in Abschnitt 5.4.1.1 diskutiert, ist eine Auflösung, welche aus der Programmdokumentation stammt, eventuell wahrscheinlicher, wenn es sich bei der Abkürzung um ein Akronym handelt. Oder umgekehrt, für ein Akronym ist eine Auflösung aus der Programmdokumentation eventuell wahrscheinlicher, als eine aus dem Quelltext. Aber auch andere Statistiken, wie die allgemeine Häufigkeit des Auftretens eines Wortes, oder die Häufigkeit des Auftretens eines Wortes in einem bestimmten Kontext, können als Gütemaß dienen.

Sollte dies nicht vorweg genommen worden sein, folgt aus der Auflösung die Entscheidung über die Erkennung und Spaltung des Bezeichners. Wurde keine Auflösung entschieden, entweder weil kein Kandidat gefunden wurde, oder kein Kandidat eine gewisse Mindestgüte erfüllt, kann dies entweder so interpretiert werden, dass der Bezeichner keine Abkürzung enthält, oder dass das Verfahren nicht in der Lage war die richtige Auflösung zu finden, oder es wird unter den gefundenen, gleichwertig besten Kandidaten einer willkürlich ausgewählt.

Ebenso folgt aus einer Auflösung, welche nicht identisch mit dem Bezeichner ist, sowohl, dass der Bezeichner abgekürzt gewesen sein muss, als auch eine Entscheidung bezüglich der Spaltung.

5.4.3 Abkürzungsauflösungsmethoden aus verwandten Arbeiten

Das Verfahren von Lawrie et al. aus dem Jahre 2007 [LFB07] verwendet Stellvertretersymbol-Erweiterung zur Generierung von Auflösungskandidaten, und nutzt einen lernenden Algorithmus zur Auswahl des besten Kandidaten, wobei für Multiwort-Bezeichner eine Methode aus der Maschinen-Übersetzung für weitere Mehrdeutigkeitsauflösung eingesetzt wird. Sie erreichen damit eine Genauigkeit von 58%. 2011 nutzen Lawrie und Binkley für ihren *Normalize* [LB11] Ansatz erneut Stellvertretersymbol-Erweiterung, und wählen den besten Kandidaten mittels einer kombinierten Gleichheitsmetrik über alle Auflösungs-Segmente des gesamten Bezeichners, wobei sie ihr Experiment mehrfach mit verschiedenen Wissensquellen und Wissensquellen-Kombinationen durchführen, um eine Aussage über die Eignung der Wissensquellen treffen zu können. Sie evaluieren ihr Verfahren auf den Quelltexten zweier Programme, mit sehr unterschiedlichem Erfolg: Bei Betrachtung der Genauigkeit bezüglich des gesamten Bezeichners erreichen sie knapp über 30% beziehungsweise knapp über 50%, bezüglich der einzelnen weichen Wörter knapp unter 50% beziehungsweise unter 66%. Insgesamt lässt ist Genauigkeit bezüglich der gesamten Bezeichner angesichts der Fortschritte in dem Gebiet aus heutiger Sicht nicht mehr vergleichbar mit anderen Verfahren. Interessant ist aber, dass die Dokumentation die allgemein nützlichste Wissensquelle zu sein scheint. Maschinelles Lernen für die Auflösung zu verwenden klingt allgemein vielversprechend für Probleme wie dieses, in denen viele Faktoren Indizien für die richtige Lösung sein könnten, aber deren Gewichtung für die Entscheidung schwer manuell einzuschätzen ist. Gleichzeitig ist es denkbar, dass die einflussreichsten dieser Faktoren sich einfacher und robuster, also mit verlässlich reproduzierbaren Ergebnisse, mit deterministischen Algorithmen und vorgefertigten Statistiken umsetzen lassen. Hier jedenfalls

schneidet das Verfahren von Lawrie et al. schlechter ab als das auf Unigramm-Grammatiken basierende Verfahren von Alatawi et al. [AXX17].

Einige Verfahren gehen davon aus, dass der richtige Auflösungskandidat am wahrscheinlichsten möglichst nahe am zu expandierenden abgekürzten Bezeichner innerhalb des Quelltextes zu finden ist. Sie priorisieren also den konkretesten Kontext, so zum Beispiel die Kommentare in unmittelbarer Nähe [HFB+08], die Parameter und Paramter-Typen einer Methode [JLZZ18], oder den Inhalt der Klasse, in welcher der Bezeichner sich befindet [AXX17] [AXY18]. Einige beschränken sich vollständig auf diesen Kontext, andere ziehen nur bei mangelndem Erfolg abstraktere Kontexte in Erwägung [HFB+08] [CDMM12], wie entfernteren Quelltext, Dokumentation oder Wörterbücher.

AMAP [HFB+08] löst Abkürzungen mittels regulärer Ausdrücke auf. Aus jeder Abkürzung werden, mit einem Regelsatz je Abkürzungstyp, verschiedene reguläre Ausdrücke erzeugt, und diese mit natürlichsprachlichen Termen aus den verschiedenen Wissensquellen abgeglichen. Auf einer Musterlösung aus 250 Bezeichnern erzielt *AMAP* eine Genauigkeit von 58,8%.

LINSEN [CDMM12] löst Abkürzungen in Quelltext-Bezeichnern auf, indem es einen Matching-Graph über den Bezeichner aufbaut, wobei die Buchstaben Knoten sind und die Pfade dazwischen mögliche Abkürzungen oder natürlichsprachliche Wörter darstellen. Ein Auflösungskandidat wird also gefunden, indem ein Pfad auf diesem Graphen Buchstaben aus dem Kandidaten in der richtigen Reihenfolge enthält. *LINSEN* wurde auf vier Programmen evaluiert, mit Musterlösungen die zwischen 211 und 957 Bezeichnern enthielten, und erzielte Genauigkeiten zwischen 56,6% und 99,1%, wobei *LINSEN* auf zwei der Musterlösungen mit *Normalize* verglichen wurde und es beide Male überbot.

Alatawi et al. [AXX17] verwenden Unigramm-Grammatiken zur Generierung und Auswahl von Auflösungskandidaten. Hierzu spalten sie den Bezeichner zunächst in weiche Wörter, wofür sie hier den synonymen Begriff ANUP einführen, sodass die Auflösung eines dieser ANUPs es jeweils auf ein Unigramm abbildet. Für die Spaltung abgekürzter Bezeichner werden zunächst alle möglichen aus ihm zu bildenden ANUPs generiert, die gewählte Spaltung wird letztlich zusammen mit dem besten Auflösungskandidaten des Bezeichners gewählt. Die Auflösungskandidaten stammen aus dem Quelltext, genauer aus der Klasse, in welcher die Abkürzung auftritt. Da auch diese nicht abgekürzten Bezeichner zusammengesetzt sein können, wird zur ihrer Spaltung ein Werkzeug namens *WordSegment 0.6.2* von Grant Jenks [Jen14] verwendet. Eine der Unigramm-Grammatiken dieser Kandidaten bezieht sich auf ihre Frequenz in einem Datensatz abseits vom Quelltext. Hier wurden dafür in der Evaluation eine Software-basierte Unigramm-Grammatik [XXA+18] und eine natürlichsprachliche Unigramm-Grammatik [BF06] gegenübergestellt. Wenig überraschend erzielte das Verfahren unter Nutzung der Software-basierten Unigramm-Grammatik die besseren Ergebnisse. Unter Einsatz dieser Unigramm-Grammatik wurde auch das Gesamtergebnis von 83,62% Genauigkeit erzielt. Interessant ist auch, dass hier, wie bei den meisten Verfahren, Akronyme und einzelne Buchstaben, welche als Sonderfall der Akronyme betrachtet werden können, am schwersten aufzulösen sind. Ein Jahr später erweiterten Alatawi et al. [AXY18] diesen Ansatz mit Bigramm-Grammatiken, indem sie die Wahrscheinlichkeit des Auftretens eines Kandidaten im Kontext des vorangehenden Wortes nutzen. Sie gehen davon aus, dass dies in der Theorie bessere Ergebnisse liefern müsste. In der Evaluation tut es das mit 78% aber nicht. Allerdings wurde kein direkter Vergleich mit der Variante vom Vorjahr auf den gleichen Daten durchgeführt, und hier wurde ein weit kleinerer Datensatz eingesetzt. Wird allerdings der von Alatawi et al. veröffentlichte Quelltext auf den mitgelieferten Beispielen ausgeführt, ist das Bigramm-basierte Verfahren, unter Verwendung der Software-basierten Uni- und Bigramm-Grammatiken, tatsächlich besser als das Unigramm-basierte.

Das Verfahren *TRIS* [GGG+12] von Guerrouj et al. wurde bereits in Abschnitt 5.3.1 genauer beschrieben. Es soll, mithilfe einer gerichteten Baumstruktur, Bezeichner sowohl spalten, als auch expandieren. Da es allerdings nur im Hinblick auf die Ergebnisse der Spaltung evaluiert wurde, ist der Vergleich mit anderen Abkürzungsauflösungsverfahren nicht quantifizierbar, weshalb es hier als Spaltverfahren analysiert wird.

Jiang et al. [JLZZ18] versuchen, mit einer ähnlichen Priorisierung der Wissensquellen wie *LINSEN* [CDMM12], ein noch präziseres Verfahren zu entwickeln, indem sie sich noch mehr auf die Auflösung von Abkürzungen in einen bestimmten Quelltextkontext beschränken, und indem sie die Besonderheiten der dort auftretenden Abkürzungen ausnutzen. Genauer versuchen sie nur Abkürzungen in Parametern aufzulösen, und priorisieren auch die Parameter selbst, beziehungsweise deren jeweilige abstrakten oder konkreten Vertreter und Typen, als Wissensquelle. Mit einer Präzision von 95% ist dieser Ansatz zwar vielversprechend, seine Grenzen stecken allerdings in seiner Definition. Zudem kann argumentiert werden, dass der *AMAP*-Ansatz [HFB+08] den größten Teil des Quelltextes, mit der Betrachtung von direktem Kontext beziehungsweise Methodenrumpf eines Bezeichners, bereits auf sehr ähnliche Art abdeckt. So gesehen bleiben also nicht viele Kontexte, die auf diese Art einzeln betrachtet werden könnten.

Allgemein scheint die Suche in einem gewissen Kontext gute Ergebnisse zu liefern, was vor allem Alatawi et al. [AXX17] zeigen. Doch die Statistiken von Newman et al. [NDA+19] zeigen, dass auch andere Wissensquellen die richtigen Auflösungen von Abkürzungen enthalten, und dass einige Abkürzungsauflösungen sogar nur einmalig in einem anderen Artefakt als dem Quelltext auftreten, insbesondere in der Sprachdokumentation. Bei Beschränkung der Kandidatensuche auf den Kontext im Quelltext können also nicht alle Abkürzungen aufgelöst werden. Zudem ist es nicht nur möglich, die richtige Auflösung nicht zu finden, sondern eine falsche Auflösung an ihrer statt. Um dies zu vermeiden, müsste von vorne herein das richtige Artefakt, beziehungsweise der richtige Kontext, betrachtet werden. Dies könnte an Eigenschaften der Abkürzung festgemacht werden, wie dem Typ oder dem Ort der Abkürzung. Doch dazu ist nicht genug darüber bekannt, wo die Auflösungen verschiedener Abkürzungen typischerweise am ehesten zu finden sind.

Das Ergebnis des Verfahrens von Alatawi et al. aus dem Jahre 2018 [AXY18] ist das beste unter den in der Metastudie von Newman et al. [NDA+19] betrachteten Verfahren. Die einzigen möglichen Konkurrenzen sind *TRIS* [GGG+12], doch ist dies unklar, da *TRIS* nur als Spaltverfahren evaluiert wurde, und das Verfahren von Jiang et al. [JLZZ18], wobei deren Verfahren sich auf Parameter beschränkt. In der Arbeit von Newman et al. werden außerdem die in Beispiel 5.4 vorgestellten Abkürzungstypen, Einwort-Abkürzungen in Form von Präfix oder weggelassene Buchstaben, Multiwort-Abkürzungen in Form von Akronymen oder aus den zuvor beschriebenen Abkürzungstypen zusammengesetzte Multiwort-Abkürzungen, als Standard vorgeschlagen. Die Studie stellt mehrere Verfahren, von denen die meisten oben diskutiert wurden, gegenüber. Allerdings wurden die Verfahren nicht nach-implementiert und auf einem einheitlichen Datensatz getestet, sondern nur ihre Ergebnisse aus ihren jeweiligen eigenen Evaluationen gegenübergestellt. Hierbei wurde bemängelt, dass nicht alle Arbeiten die gleichen, beziehungsweise alle denkbaren Metriken angegeben haben, und zudem nur wenige ihre Ergebnisse nach verschiedenen Gegebenheiten, wie den Einfluss von Abkürzungstypen, aufgeschlüsselt haben. Auch Aufschlüsselung nach ganzem Bezeichner oder einzelnem weichen Wort kann interessant sein, darauf geht die Arbeit allerdings nicht ein.

Neben der Evaluation von Abkürzungsauflösungsverfahren präsentieren Newman et al. auch Statistiken bezüglich dem Auftreten von Abkürzungen in verschiedenen Programm-Quelltexten, die Häufigkeit gewisser Abkürzungstypen in diesen, und das Auftreten der korrekten Auflösungskandidaten in verschiedenen Wissensquellen. Dabei ist besonders die

Häufigkeit des richtigen Kandidaten in der Sprachdokumentation, dicht gefolgt von der Projektdokumentation hervorzuheben, wobei besonders in der Sprachdokumentation richtige Kandidaten vorkommen, welche ausschließlich dort zu finden sind. Auch sind Präfix-Abkürzungen unangefochten die häufigsten Abkürzungen in Quelltext, gefolgt von Wörtern, in welchen Buchstaben weggelassen wurden. Akronyme liegen deutlich dahinter. Es wurden so gut wie keine kombinierten Multiwort-Bezeichner gefunden. Letzteres erscheint kontraintuitiv, und so erscheint es, obwohl die Forscher fünf Projekte betrachtet haben, wie ein Stichprobenfehler, oder als hätte der Algorithmus zur Suche nach den Abkürzungstypen versagt, obwohl sein Ergebnis manuell kontrolliert wurde.

Ratinov und Gudes versuchen in „Abbreviation Expansion in Schema Matching and Web Integration“ Abkürzungen nicht in Quelltext, sondern in Datenbanken aufzulösen [RG04]. Sie suchen nach Kandidaten, indem sie Abkürzungsmuster wie eine Maske über natürlichsprachliche Wörter legen, um abzuwägen ob die Abkürzung durch Weglassen von Buchstaben in diesem Wort hätte gebildet werden können. Diese Methode wenden sie sowohl in einer deterministischen Heuristik, als auch in einem neuronalen Netz an. Leider wurde bei der Evaluation die Güte der Verfahren nicht quantifiziert, weshalb es schwer abzuschätzen ist wie vielversprechend dieser Ansatz ist.

Zhang et al. haben in „Entity Linking with Effective Acronym Expansion, Instance Selection and Topic Modeling“ [ZSST11] das Ziel, Akronyme in natürlichsprachlichem Text aufzulösen, um Begriffe aus Fließtext mit Wissensdatenbanken wie Enzyklopädien zu verknüpfen. Es wird davon ausgegangen, dass Akronyme immer als vollständig groß geschriebene Wörter vorliegen. Zunächst wird gehofft, dass die Abkürzung in Klammern nach ihrer Auflösung im Text steht, oder umgekehrt. Wenn nicht, wird die Auflösung einer Abkürzung anderswo im selben Text gesucht, indem Token mit passendem Anfangsbuchstaben und entsprechenden Nachfolge-Token gesucht werden. Bei jedem solchen Textabschnitt werden mehrere Kandidaten erzeugt, indem verschiedene Möglichkeiten für Anfang und Ende des Begriffs und Einbeziehen und Nichteinbeziehen von Stoppwörtern berücksichtigt werden. Die so entstandene große Menge an Kandidaten wird mit einem überwachten Lernalgorithmus auf als valide betrachtete Kandidaten beschränkt, von welchen letztendlich ein Klassifizierer den besten auswählt. Da die von Zhang et al. behandelte Domäne nicht der Quelltext eines Programms ist, und der einzig betrachtete Abkürzungstyp Akronyme sind, lassen sich die Methoden und Ergebnisse nicht einfach auf das Problem der allgemeinen Abkürzungsauflösung in Quelltext übertragen. Aber dennoch ist eine Genauigkeit von 92,9% in der Akronymauflösung verhältnismäßig hoch, und es wäre möglich, dass einige Teile der Methode sich an andere Domänen anpassen ließen. Außerdem sind Akronyme eine Schwachstelle vieler Abkürzungsauflösungsverfahren für Quelltext, ein auf Akronyme spezialisiertes Verfahren als Ergänzung zu entwickeln könnte dem Abhilfe verschaffen.

Liu et al. behandeln in „Exploiting Task-Oriented Resources to Learn Word Embeddings for Clinical Abbreviation Expansion“ [LGM+15] das Problem der Auflösung von Abkürzungen in Notizen in der Intensiv-Pflege. Abkürzungen werden mittels regulärer Ausdrücke im Text gefunden. Mit dem Vokabular aus den Notizen, Wikipedia-Artikeln und Fachliteratur wie Artikeln, Journalen und Büchern werden Worteinbettungen trainiert, die später der Bewertung von Kandidaten dienen sollen. Die Kandidaten selbst werden aus einer Liste medizinischer Abkürzungen von *allacronyms.com* gewonnen. Die Genauigkeit von 82,27% bewegt sich zwar unter den vielversprechenderen Verfahren, das Verfahren lässt sich aber nicht direkt auf Quelltext übertragen.

Leider sind nicht alle der hier aufgelisteten Verfahren dergestalt evaluiert worden, dass sie quantitativ miteinander vergleichbar wären. Zudem erscheinen einige der Methoden, welche mit verhältnismäßig geringem Erfolg oder geringer Aussagekraft evaluiert wurden, dennoch plausibel, und es kann nicht unbedingt davon ausgegangen werden, dass die Grun-

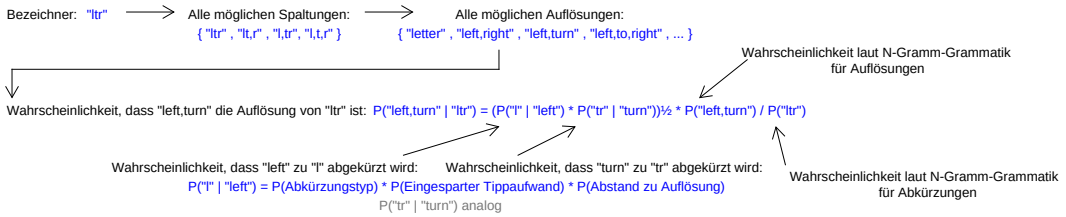

Abbildung 5.2: Ablauf der Auflösung eines Bezeichners bei Alatawi et al.

didee, aufgrund eines einzelnen Verfahrens, welches wenig erfolgreich oder mit der Zeit überholt worden ist, verworfen werden sollte. Beispiele hierfür sind die Verwendung von maschinellem Lernen zur Auswahl des besten Auflösungskandidaten durch Lawrie et al. [LFB07], oder die Kandidatensuche mit einem Abkürzungsmuster von Ratinov und Gudes [RG04]. Umgekehrt liefert die Beschränkung auf einen gewissen Kontext für die Kandidatensuche, wie bei Alatawi et al. [AXX17] [AXY18] und Jiang et al. [JLZZ18] die bisher besten Ergebnisse, doch beschränkt laut den Statistiken von Newman et al. [NDA+19] zufolge unweigerlich die Genauigkeit des Verfahrens, da einige Abkürzungen so schlichtweg nicht richtig aufgelöst werden können. Zudem gibt es noch einige Ansätze, welche noch überhaupt nicht verfolgt wurden. Doch sind der Ansatz der Nutzung von N-Gramm-Grammatiken nach Alatawi et al. [AXX17] [AXY18], und der Ansatz der sehr präzisen Auswahl eines bestimmten Kontextes zu Auflösung bestimmter Abkürzungen, wie das Parameter-fokussierte Verfahren von Jiang et al. [JLZZ18], die bislang effektivsten, und sind somit die vielversprechensten Ausgangspunkte zur weiteren Forschung.

5.4.4 Weitere Methoden zur Auflösung von Abkürzungen in Quelltext

Einige der Ansätze aus verwandten Arbeiten zur Auflösung von Abkürzungen bieten weiteres Potenzial, welches bislang noch nicht vollends ausgeschöpft wurde. Zudem werden hier weitere Ansätze vorgestellt, welche in den in Abschnitt 5.4.3 und Kapitel 4 vorgestellten Verfahren noch gar keine Erwähnung gefunden haben.

5.4.4.1 N-Gramme

Die Verfahren von Alatawi et al. [AXX17] [AXY18] liefern unter allen verwandten Arbeiten mit die besten Ergebnisse, und bieten Potential zum Fortsetzen der Idee. Nach dem Einsatz von Uni- und Bigrammen sind Trigramme der konsequente nächste Schritt, um hoffentlich mit der Berücksichtigung noch tieferer statistischer Zusammenhänge zwischen verschiedenen Wörter auch besser einschätzen zu können, ob diese wahrscheinlich in einem zusammengesetzten Bezeichner auftreten. Zunächst werden alle möglichen Spaltungen eines Quelltextbezeichners bestimmt, dann daraus alle möglichen Auflösungskandidaten.

$$P(ZK|B) = (\prod_{i=1}^{n} P(A_i|K_i))^{1/n} * P(ZK)/P(B) \tag{5.1}$$

Die Gleichung 5.1 zeigt die Wahrscheinlichkeit $P(ZK|B)$, dass ein aus n Segmenten zusammengesetzter Kandidat ZK die richtige Auflösung eines Quelltextbezeichners B ist. Sie setzt sich zusammen aus dem normierten Produkt der Wahrscheinlichkeiten $P(A|K)$, dass die Segmente des zusammengesetzten Auflösungskandidaten, also die einzelnen Kandidaten K, zu den abgekürzten Token A aus dem Quelltextbezeichner abgekürzt würden, und

den Wahrscheinlichkeiten *P(ZK)* und *P(B)*, dass der zusammengesetzte Kandidat *ZK* in einem Text und der Quelltextbezeichner *B* in einem Quelltext auftreten.

$$P(A|K) = P_{Typ} * P_{TES} * P_D \tag{5.2}$$

Die Wahrscheinlichkeit, dass eine Auflösung zu der entsprechenden Abkürzung aus dem Quelltextbezeichner abgekürzt würde, setzt sich zusammen aus verschiedenen Wahrscheinlichkeiten, welche mittels Statistiken aus einer Studie von Xu et al. [XXA+18] bestimmt werden. Die Studie von Xu et al. „Statistical Unigram Analysis for Source Code Repository“ wertet verschiedene statistische Eigenschaften bezüglich des Auftretens von Quelltextbezeichnern in Software aus. Dabei werden auch Statistiken darüber aufgestellt, wie wahrscheinlich Präfix-Abkürzungen oder Abkürzungen mit weggelassenen Buchstaben sind, wie wahrscheinlich welches Maß an eingespartem Tippaufwand einer Abkürzung gegenüber ihrer Auflösung ist, und wie wahrscheinlich welche Entfernung, gemessen in Quelltextzeilen, zwischen einer Abkürzung und der nächsten Erwähnung ihrer Auflösung im Quelltext ist. Eben diese Statistiken werden genutzt, um die Wahrscheinlichkeit einer bestimmten Auflösung für eine bestimmte Abkürzung aus einem Quelltextbezeichner im jeweiligen Kontext zu bestimmen. Die Wahrscheinlichkeit *P(A|K)*, dass ein Kandidat *K* zu der Abkürzung *A* abgekürzt würde, wird in Gleichung 5.2 gezeigt. Sie ist das Produkt der Wahrscheinlichkeit P_{Typ} des Abkürzungstyps, der Wahrscheinlichkeit P_{TES} des eingesparten Tippaufwandes, und der Wahrscheinlichkeit P_D der Distanz zwischen der Abkürzung und dem Kandidaten im Quelltext.

Die Wahrscheinlichkeit für das Auftreten des Auflösungskandidaten an sich, und die Wahrscheinlichkeit des Auftretens des abgekürzten Bezeichners, wird mittels der N-Gramm-Grammatik des jeweiligen Verfahrens bestimmt. Bei Alatawi et al. sind dies in einem Verfahren Unigramm-Grammatiken [AXX17], im anderen Verfahren Bigramm- und Unigramm-Grammatiken in Kombination [AXY18]. Der Ablauf der Bestimmung und Bewertung eines Abkürzungsauflösungskandidaten wird in Abbildung 5.2 grafisch anhand eines Beispiels illustriert.

$$P_{uni}(ZK) = (\prod_{i=1}^{n} P(K_i))^{1/n} \tag{5.3}$$

$$P_{uni}(B) = (\prod_{i=1}^{n} P(A_i))^{1/n} \tag{5.4}$$

Beispiel 5.15: Unigramm-basierte Auftretenswahrscheinlichkeit eines Auflösungskandidaten

P(„left,to,right“) = (P(„left“) * P(„to“) * P(„right“))$^{1/3}$

Wobei beispielsweise *P(„left“)* in der Unigramm-Grammatik nachgeschlagen wird.

Bei Verwendung des Unigramm-basierten Verfahrens wird, wie in Beispiel 5.15 gezeigt, die Auftretenswahrscheinlichkeit des Auflösungskandidaten aus Gleichung 5.3, beziehungsweise Auftretenswahrscheinlichkeit des Quelltextbezeichners aus Gleichung 5.4, für jedes Token oder Segment einzeln aus der Statistik ausgelesen, multipliziert und das Produkt nach der Länge des Auflösungskandidaten normiert.

$$P_{bi}(ZK) = (P(K_1) * [(P(K_1, K_2)/P(K_1)) * ... * (P(K_{n-1}, K_n)/P(K_{n-1}))])^{1/n} \quad (5.5)$$

$$P_{bi}(B) = (P(A_1) * [(P(A_1, A_2)/P(A_1)) * ... * (P(A_{n-1}, A_n)/P(A_{n-1}))])^{1/n} \quad (5.6)$$

Beispiel 5.16: Bigramm-basierte Auftretenswahrscheinlichkeit eines Auflösungskandidaten

P(„left,to,right“) = (P(„left“) * (P(„left to“) / P(„left“)) * (P(„to right“) / P(„to“))$^{1/3}$

Wobei beispielsweise *P(„left to“)* in der Bigramm-Grammatik und *P(„left“)* in der Unigramm-Grammatik nachgeschlagen wird.

Beim Bigramm-basierten Verfahren wird die Wahrscheinlichkeit des Auflösungskandidaten, beziehungsweise des zusammengesetzten Quelltextbezeichners, paarweise aus den aufeinander folgenden Token im Auflösungskandidaten, beziehungsweise aus den Segmenten im zusammengesetzten Quelltextbezeichner, aus der Bigramm-Grammatik gewonnen, und bezüglich der Unigramm-Wahrscheinlichkeit der einzelnen Token, beziehungsweise Segmente, bedingt, wie in Gleichung 5.5 für Auflösungskandidaten und Gleichung 5.6 für Quelltextbezeichner zu sehen ist, und in Beispiel 5.16 an einem Beispiel gezeigt wird. Mittels der Wahrscheinlichkeit des Zusammengesetzten Auflösungskandidaten aus Gleichung 5.1, für jeden zusammengesetzten Auflösungskandidaten, wird der, welcher für richtig befunden wird, ausgewählt.

Zudem lässt die originale Implementierung von Alatawi et al. beim Verfahren für Bigramm-basierte Auflösung keine Auflösungskandidaten zu, welche nicht aus mindestens aus zwei Token zusammengesetzt sind. Dies ist zwar im Bezug auf die Musterlösung legitim, da dort tatsächlich alle Abkürzungen aus mindestens zwei Token zusammengesetzt sind, lässt sich aber nicht auf alle Anwendungsfälle verallgemeinern. Für Kandidaten, welche aus nur einem Token bestehen, auf die Unigramm-basierte Wahrscheinlichkeit zurückzugreifen, ist leider keine Option, da die Unigramm-basierten Wahrscheinlichkeiten allgemein als höher bewertet werden. Beim Vergleich der Wahrscheinlichkeit der durch das Verfahren als richtig befundenen Auflösungskandidaten, bestimmt durch Debugging des originalen Quelltextes von Alatawi et al., ist der durchschnittliche Unterschied zwischen den Wahrscheinlichkeiten, die mittels des Uni- oder Bigramm-basierten Ansatzes bestimmt wurden, ein Faktor von über 6.000 beziehungsweise von über 10.000, je nach dem ob die Uni- und Bigramm-Grammatiken aus den Software-basierten Quellen [XXA+18] oder aus den natürlichsprachlichen Quellen [BF06] verwendet werden. Die Unigramm-basierte Wahrscheinlichkeit für Auflösungskandidaten zu verwenden, die aus nur einem Token bestehen, würde also dazu führen, dass immer einer dieser aus nur einem Token bestehenden Auflösungskandidaten ausgewählt wird, und die eventuell richtigen zusammengesetzten Kandidaten ignoriert werden.

5.4.4.2 Akronyme

Viele der Verfahren zur Auflösung von Abkürzungen in Quelltext scheinen besondere Defizite bei der Auflösung von Akronymen zu haben. Gleichzeitig sind Akronyme außerhalb des Quelltextes die mutmaßlich am häufigsten eingesetzten Abkürzungen. Abkürzungsauflösungsverfahren aus anderen Domänen explizit für diesen Abkürzungstyp zu übernehmen, oder anderweitig ein Werkzeug zu entwickeln, welches gezielt Akronyme aus Quelltext

auflöst, könnte somit einer konkreten Schwachstelle der bisherigen Bemühungen entgegenwirken. In Abschnitt 5.2 wurde die Vermutung aufgestellt, dass Akronyme häufiger aus der Domäne stammen, als spontan im Quelltext erzeugt worden zu sein. Sollte dies zutreffen, könnte zudem gezielt in Glossaren der Dokumentation nach ihnen und einer entsprechenden Auflösung gesucht werden. Außerdem könnte im Fließtext danach gesucht werden, ob dort nicht in unmittelbarer Nähe des Akronyms, zum Beispiel in Klammern dahinter, eine Erklärung oder Auflösung für dieses Akronym vorliegt. Allgemein könnte ein spezifisches Verfahren zum Auflösen von Akronymen, ähnlich dem Vorschlag von Jiang et al. [JLZZ18] bezüglich ihres auf einen bestimmten Quelltext-Kontext beschränkten Verfahrens, in Kombination mit einem oder mehreren anderen Verfahren zur Auflösung von Abkürzungen in Quelltext eingesetzt werden.

5.4.4.3 Worteinbettungen

Die Idee von Liu et al. [LGM+15], Worteinbettungen für die Abkürzungsauflösung einzusetzen, ist angesichts des Ziels, die semantische Intention, welche der Entwickler bei der Formulierung eines abgekürzten Bezeichners hatte, zurückzugewinnen, sehr naheliegend. In Abschnitt 5.2 wird darauf eingegangen, dass ein Zusammenhang zwischen abgekürzten Bezeichnern und natürlichsprachlichen Begriffen besteht. Die in einem Worteinbettungsmodell festgehaltene Information über den Zusammenhang verschiedener Terme kann dazu verhelfen, die Wahrscheinlichkeit eines Auflösungskandidaten, im Kontext des umliegenden Quelltextes, oder der Auflösungskandidaten anderer Teilbegriffe des gleichen Bezeichners, zu ermitteln. Ein Auflösungskandidat, welcher im Sinne des Worteinbettungsmodells dem Kontext ähnlicher ist als ein anderer, ist auch mit größerer Wahrscheinlichkeit der richtige. Worteinbettungsmodelle können also der Auswahl zur Ermittlung des richtigen Kandidaten aus einer Vorauswahl plausibler Kandidaten dienen. Dennoch scheinen Worteinbettungen innerhalb der Domäne der Auflösung von Abkürzungen aus Quelltext noch nicht eingesetzt worden zu sein.

Hier gibt es mehrere Herangehensweisen, welche in Betracht gezogen werden können. Zunächst stellt sich die Frage, ob ein vorgefertigtes Worteinbettungsmodell eingesetzt wird, oder ein neues auf bestimmten Wissensquellen, zum Beispiel dem betrachteten Projekt, das heißt seinem Quelltext und seiner Dokumentation, trainiert wird. Bei beidem stellt sich die Frage, wie spezialisiert die Artefakte sein sollten, auf welchen eine Einbettung trainiert wird oder wurde. Wird ein neues Worteinbettungsmodell trainiert, muss die Gesamtgröße der verwendeten Dokumente ausreichen, um ein Modell zu erzeugen, welches die Wahrscheinlichkeit des gemeinsamen Auftretens aller Begriffe, welche in den richtigen Auflösungen vorkommen, angemessen wider zu spiegeln. Eine zu kleine Wissensquelle für das Training könnte nicht alle gewünschten Begriffe enthalten, oder einen Stichprobenfehler in der beobachteten Verteilung darstellen. Eine zu große Wissensquelle könnte wiederum ein derart allgemeines Modell erzeugen, dass dies dem Ziel, ein gezielt für das betrachtete Projekt geeignetes Modell zu erzeugen, widersprechen würde. Wird sich beispielsweise dafür entschieden, ein Worteinbettungsmodell auf Quelltext zu trainieren, wäre der betrachtete Quelltext selbst vielleicht zu klein. Um dem entgegen zu wirken, könnte eine Menge von mehreren Programmen dazu verwendet werden. Um hierbei nicht dem Ziel eines spezialisierten Modells zuwider zu handeln, könnten gezielt Programme aus der gleichen Domäne verwendet werden. Im anderen Fall, der Nutzung eines zuvor trainierten Worteinbettungsmodells, ist zu beachten, auf was für Datensätzen dieses Modell trainiert wurde. Ähnlich wie die unterschiedlichen Unigramm-Grammatiken die Alatawi et al. [AXX17] verwenden, könnten Einbettungen auf Wissensquellen aus natürlicher Sprache oder aus der Software-Domäne trainiert werden. Bei Alatawi et al. liefern die N-Gramm-Grammatiken aus der Software-Domäne die besseren Ergebnisse, was erwartungsgemäß auch für Einbettungen der Fall sein sollte. Denn wenn ein Worteinbettungsmodell auf einem Software-bezogenen

Datensatz, oder auf einem Datensatz aus der Domäne des betrachteten Quelltextes trainiert wurde, ist zu erwarten, dass die Zusammenhänge zwischen den Wortvektoren besser die konkrete Semantik der Domäne widerspiegeln, als bei einem allgemeineren Worteinbettungsmodell, das Zusammenhänge herstellt, welche hier nicht relevant sind. Andererseits könnte eine auf Quelltext trainierte Einbettung auch Bezeichner beinhalten, welche selbst abgekürzt oder in anderer Weise keine natürlichen Wörter sind. Für die Einschätzung eines Auflösungskandidaten, welcher im Sinne des Worteinbettungsmodells einem solchen nicht natürlichen Wort ähnelt, wäre dies kein Problem. Wird jedoch ein solches Wort durch das Modell als Auflösungskandidat einer Abkürzung für plausibel erachtet, würde das dem Zweck des Verfahrens widersprechen, Abkürzungen zu natürlichen Wörtern aufzulösen. Ein Worteinbettungsmodell, welches auf natürlicher Sprache trainiert wurde, könnte wiederum ein größeres Vokabular abdecken, da es sich nicht auf Begriffe aus der Domäne beschränkt. Somit könnten daraus auch Auflösungskandidaten für wahrscheinlich erachtet werden, welche eigentlich in der Domäne des Quelltextes unwahrscheinlich sind.

Es gibt verschiedene Worteinbettungsverfahren, welche sowohl für die Auswahl eines vorgefertigten Datensatzes, als auch für das Trainieren eines neuen Worteinbettungsmodells in Frage kommen. *GloVe* [PSM14] beispielsweise steht für „global vectors“ (globale Vektoren), und trainiert dem Namen entsprechend die Statistik des gemeinsamen Auftretens zweier Wörter auf dem gesamten Datensatz. *word2vec* [MCCD13] hingegen beobachtet das gemeinsame Auftreten innerhalb des lokalen Kontextes der benachbarten Wörter. Da bei der Auflösung von Abkürzungen aus Quelltext nur Begriffe einer bestimmten Domäne in Frage kommen, und außerdem die einzelnen Bezeichner eine bestimmte Rolle in einem Prozessablauf in der Domäne spielen, kann von der Nutzung eines auf *word2vec* anstelle auf *GloVe* trainierten Modells erhofft werden, dass es, auf Grund der Nutzung des lokalen Kontextes zur Beobachtung des gemeinsamen Auftretens, die zurück zu verfolgenden semantischen Zusammenhänge besser widerspiegelt. Insbesondere beim Trainieren eines neuen Worteinbettungsmodells auf dem Quelltext oder anderen Software-Artefakten könnte der beschränkte Fokus von *word2vec* eine interessante Wechselwirkung mit dem Ansatz, im unmittelbaren Kontext eines abgekürzten Bezeichners im Quelltext nach einer Auflösung zu suchen, ergeben. Das Worteinbettungsverfahren *fastText* [BGJM17] hat die Besonderheit, dass es Subwortinformationen berücksichtigt. In *fastText* werden Terme als eine Menge von N-Grammen dargestellt, sodass nicht nur die Verhältnisse des Auftretens einzelner Terme, sondern auch von deren Segmenten berücksichtigt werden. Da bei der Auflösung von Abkürzungen aus Quelltextbezeichnern auch zusammengesetzte Bezeichner betrachtet werden, könnte gerade ein neues, auf dem Quelltext trainiertes *fastText*-Modell Vorteile gegenüber anderen Worteinbettungsverfahren haben, es die Zusammenhänge der Wörter innerhalb zusammengesetzter Bezeichner modellieren würde, und nicht nur die zwischen den Bezeichnern. Aber auch auf einem im Voraus trainierten Modell könnte es von Vorteil sein, die Segmente eines Auflösungskandidaten bei der Ermittlung der Ähnlichkeit des Kandidaten zum Kontext zu berücksichtigen.

Um den Zusammenhang zweier Terme mit Hilfe eines Worteinbettungsmodells zu quantifizieren, wird die Cosinusähnlichkeit zwischen den entsprechenden Vektoren ermittelt. Eben solche Zusammenhänge sollten auch Aufschluss darüber geben, welcher Kandidat aus einer Auswahl an Auflösungskandidaten am wahrscheinlichsten der richtige ist. Die Vektoren, zu denen hierfür die Ähnlichkeit ermittelt werden kann, sind die der anderen Wörter im Kontext, oder die Auflösungskandidaten der anderen Segmente eines zusammengesetzten Quelltextbezeichners, beziehungsweise die einzelnen Wörter eines zusammengesetzten Auflösungskandidaten. Es gibt verschiedene Möglichkeiten, wie die Wortvektoren hier eingesetzt werden können. Ein naiver Ansatz wäre, für jedes Wort in einem zusammengesetzten Auflösungskandidaten, die normalisierte Ähnlichkeit zu jedem Wort im Kontext und zu jedem anderen Wort im zusammengesetzten Auflösungskandidaten zu bestimmen. Allerdings

ist die Ähnlichkeit zu den anderen Token im gleichen Auflösungskandidaten vielleicht eher störend, da zusammengesetzte Bezeichner bewusst zusammengesetzt werden, um mehrere verschiedene Sachverhalte zu beschreiben. Ebenso muss nicht zwingend jedes Token aus dem zusammengesetzten Auflösungskandidaten jedem Token aus dem Kontext ähneln, da ja wiederum jeder Teilbegriff eines zusammengesetzten Begriffes einen anderen Sachverhalt im Kontext beschreiben könnte. Es könnte also aussagekräftiger sein, für jedes Token aus einem zusammengesetzten Auflösungskandidaten nur die Nähe zu einem gewissen Anteil des Kontextes zu betrachten, nämlich den, dem das Token am meisten ähnelt. Anstelle die Ähnlichkeit der einzelnen Token aus dem zusammengesetzten Auflösungskandidaten zum Kontext zu ermitteln, könnte auch ein normierter summierter Vektor für jeweils den zusammengesetzten Auflösungskandidaten und den Kontext gebildet werden, und deren Ähnlichkeit bestimmt werden. Jeder dieser Vektoren sollte die Semantik des Kontextes, beziehungsweise des Auflösungskandidaten, im Sinne des Worteinbettungsmodells zusammenfassen, und somit könnte ihr Vergleich eine bessere Aussagekraft darüber haben, ob der zusammengesetzte Auflösungskandidat insgesamt einen wahrscheinlichen Begriff in diesem Kontext darstellt. Eine weitere Möglichkeit, die Präzision eines Auflösungsverfahrens zu erhöhen, könnte sein, die Wortart der Auflösungskandidaten zu berücksichtigen. Im Zusammenhang mit Worteinbettungen könnten beispielsweise die Wörter aus einem zusammengesetzten Auflösungskandidaten nur mit Wörtern in den Kontext gesetzt werden, die die gleiche Wortart darstellen, um so hoffentlich den Zusammenhang zwischen dem Bezeichner und seinem Kontext besser einzufangen. Auch könnte, anstatt paarweise die Wortvektoren einzelner Token miteinander zu vergleichen, je ein Vektor aus der normierten Summe der Wortvektoren aus dem Auflösungskandidaten, beziehungsweise aus dem Kontext, erzeugt werden. Somit würde ein solcher Vektor den gesamten Auflösungskandidaten repräsentieren, und der andere den gesamten Kontext, sodass Kandidat und Kontext in Form dieser Vektoren vollständig verglichen werden können.

6 Entwurf

Als Vergleichswert und Grundlage für darauf aufbauende Verfahren, werden zunächst die Verfahren von Alatawi et al. [AXX17] [AXY18] verwendet. Diese Verfahren wurden zum einen ausgewählt, da sie, wie in Abschnitt 5.4.3 erörtert, beispielhaft für den aktuellen Stand der Forschung im Gebiet der Auflösung von Abkürzungen in Quelltext sind. Das einzige Verfahren, dem vergleichbare Ergebnisse zugemessen werden könnten, ist *TRIS* [GGG+12]. Dieses wurde allerdings nur gegen Verfahren zur Spaltung von Abkürzungen evaluiert. Somit lässt es sich leider schwerer mit anderen Verfahren zur Abkürzungsauflösung vergleichen. Zum anderen da auf ihnen aufbauend ein Trigramm-basiertes Verfahren entwickelt wird. Wie in Abschnitt 5.4.4.1 angeregt, sind Trigramm-Grammatiken in der Auflösung von Abkürzungen in Quelltext der nächste konsequente Schritt nach den Erfolgen von Alatawi et al. bei der Auflösung von Abkürzungen in Quelltextbezeichnern unter Verwendung von Uni. und Bigramm-Grammatiken.

6.1 Trigramm-Grammatiken für die Auftretenswahrscheinlichkeit von Auflösungskandidaten und Quelltextbezeichnern

Die beiden Verfahren von Alatawi et al. haben den gleichen Aufbau, sie setzen nur an einer Schnittstelle entweder Unigramm-Grammatiken, oder Uni- und Bigramm-Grammatiken in Kombination ein. Wie in Abschnitt 5.4.4.1 und noch konkreter in Abbildung 5.2 beschrieben wird, bestimmen die Verfahren von Alatawi et al. die Wahrscheinlichkeit eines Auflösungskandidaten für eine Abkürzung mit Hilfe einer Formel, in der an einer bestimmten Schnittstelle die Uni- beziehungsweise Bigramm-Wahrscheinlichkeiten des Token im Auflösungskandidaten eine tragende Rollen spielen. Dieses Verfahren wird dadurch erweitert, dass die Uni- oder Bigramm-basierten Auftretenswahrscheinlichkeiten des Auflösungskandidaten und der Abkürzung, die in Gleichung 5.3, Gleichung 5.4, Gleichung 5.5 und Gleichung 5.6 beschrieben werden, durch Trigramm-basierte Auftretenswahrscheinlichkeiten ersetzt werden.

$$P_{tri}(ZK) = (P(K_1)*[(P(K_1,K_2,K_3)/P_{bi}(K_1,K_2))*...*(P(K_{n-2},K_{n-1},K_n)/P_{bi}(K_{n-2},K_{n-1}))])^{1/n} \tag{6.1}$$

$$P_{tri}(B) = (P(A_1)*[(P(A_1,A_2,A_3)/P_{bi}(A_1,A2))*...*(P(A_{n-2},A_{n-1},A_n)/P_{bi}(A_{n-2},A_{n-1}))])^{1/n} \tag{6.2}$$

Beispiel 6.1: Trigramm-basierte Auftretenswahrscheinlichkeit eines Auflösungskandidaten

P(„abbreviation,expansion,using,trigrams“) =
(P(„abbreviation“) * (P(„abbreviation expansion using“) / P(„abbreviation,expansion“))
* (P(„expansion using trigrams“) / P(„expansion,using“))$^{1/4}$

Wobei beispielsweise *P(„abbreviation expansion using“)* in der Trigramm-Grammatik nachgeschlagen wird, und *P(„abbreviation,expansion“)* als bedingte Bigramm-Wahrscheinlichkeit, wie sie in Gleichung 5.5, beziehungsweise Beispiel 5.16 berechnet wird.

In Abschnitt 5.4.4.1 wurde bereits darauf eingegangen, wie Unigramm- und Bigramm-Grammatiken in den Verfahren von Alatawi et al. genutzt werden, um die Auftretenswahrscheinlichkeiten der Auflösungskandidaten und der Abkürzungen zu bestimmen. Um Trigramme hierfür einsetzen zu können, werden die hintereinander stehenden Token nicht nur paarweise, sondern im Dreibund betrachtet. Die Wahrscheinlichkeit von drei hintereinander auftretenden Token wird aus der Statistik ausgelesen, und bedingt nach der Wahrscheinlichkeit der darin auftretenden Bigramme, die wiederum durch die Wahrscheinlichkeit der Unigramme bedingt werden. Die gemeinsame Wahrscheinlichkeit der aneinander gereihten Trigramme ergibt die Auftretenswahrscheinlichkeit des gesamten Auflösungskandidaten oder Quelltextbezeichners. Die Berechnung dieser Trigramm-basierten Auftretenswahrscheinlichkeiten wird in Gleichung 6.1 für Auflösungskandidaten und in Gleichung 6.2 für Quelltextbezeichner beschrieben, und in Beispiel 6.1 an einem Beispiel verdeutlicht.

Um mit Bezeichnern umzugehen, welche nur zu Kandidaten aufgelöst werden können, welche aus nur ein oder zwei Elementen bestehen, werden diese mittels des Unigramm-, beziehungsweise Bigramm-basierten Verfahrens aufgelöst. Für den Fall, dass für einen Bezeichner der richtige Auflösungskandidat aus nur einem oder zwei Token besteht, aber in der Kandidatensuche auch solche Kandidaten gefunden werden, welche aus mehr Token bestehen, besteht das analoge Problem wie beim Bigramm-basierten Verfahren, welches in Abschnitt 5.4.4.1 beschrieben wurde, dass zwangsläufig ein Kandidat ausgewählt wird, welcher aus mindestens drei Token besteht.

6.2 Worteinbettungen bei der Abkürzungsauflösung

In einem anderen im Rahmen dieser Arbeit umgesetzten Verfahren werden Worteinbettungsmodelle genutzt, um die Entscheidung des richtigen Auflösungskandidaten zu treffen. *word2vec* und *fastText* wurden als Worteinbettungsverfahren ausgewählt, da *word2vec* aufgrund des Trainings auf lokalem Kontext für diesen Verwendungszweck als viel versprechender als *GloVe* eingeschätzt wird, und um die Hypthese zu prüfen, dass die Subwortinformationen, welche *fastText* verwendet, einen Vorteil für ein Abkürzungsauflösungsverfahren auf Quelltext bieten.

6.2.1 Worteinbettungsmodelle

Sowohl für die Verwendung von *word2vec*, als auch die von *fastText*, werden sowohl im Voraus trainierte, als auch explizit für diese Arbeit trainierte Worteinbettungsmodelle eingesetzt.

Dies dient, wie in Abschnitt 5.4.4.3 begründet wird, der Einschätzung, ob ein möglichst allgemeines, in einem großen Korpus trainiertes Worteinbettungsmodell, oder ein, für die Abkürzungsauflösung in einem bestimmten Quelltext, spezialisiertes Worteinbettungsmodell

geeigneter für die hier vorgestellten worteinbettungsbasierten Abkürzungsauflösungsverfahren ist. Auf Grund der Beobachtungen von Alatawi et al. [AXX17] [AXY18] bezüglich Software-basierter und nicht Software-basierter N-Gramm-Grammatiken, wird dennoch auch für allgemeinere Worteinbettungsmodelle davon ausgegangen, dass solche, welche auf einem Software-bezogenen Korpus trainiert wurden, besser geeignet sein sollten. Auf welchen Korpora Worteinbettungsmodelle, welche auf bestimmte Quelltexte spezialisiert sind, trainiert werden könnten, und welche Vor- und Nachteile diese Ansätze jeweils bieten könnten, wird in Abschnitt 5.4.4.3 diskutiert. Als naiver und einfach umzusetzender Ausgangspunkt werden solche eingesetzt, welche unmittelbar auf dem betrachteten Quelltext trainiert wurden. Dies bedeutet offensichtlich eine deutliche Spezialisierung auf den entsprechenden Quelltext, und sollte dies zu guten Ergebnissen führen, wäre das Verfahren mit sehr wenig Vorbereitung auf verschiedenen Quelltexten einsetzbar. Dieser Ansatz beschränkt den Korpus aber stärker, als vermutlich förderlich ist. Deswegen wird auch der Ansatz verfolgt, ein Worteinbettungsmodell auf dem Quelltext mehrerer Programme zu trainieren, welche aus der gleichen Domäne stammen, wie der Quelltext, in welche Abkürzungen aufgelöst werden sollen. In beiden Fällen könnte solch ein Modell aber dazu führen, dass nicht natürliche Terme als Auflösungskandidaten ausgewählten werden. Deswegen wird außerdem, beispielhaft für Domänen-Literatur, ein Worteinbettungsmodell auf einem Wikipedia-Artikel, bezüglich der Domäne eines der betrachteten Quelltexte, trainiert. Die Hoffnung hinter diesem Ansatz ist, dass dieser Korpus einerseits nur natürlichsprachliche Terme enthält, aber andererseits spezialisiert genug ist, damit keine in dieser Domäne unwahrscheinlichen Auflösungskandidaten ausgewählt werden.

6.2.2 Auflösungskandidaten

Um einen Auflösungskandidaten auszuwählen, welcher als richtig erachtet wird, muss zunächst eine Liste an Kandidaten, welche miteinander verglichen werden können, vorliegen. Dazu wird der Unigramm-basierte Auflösungsansatz [AXX17] von Alatawi et al. verwendet. Hierzu werden zwei verschiedene Herangehensweisen getestet.

Um eine möglichst unabhängige Betrachtung von Worteinbettungsmodellen als Grundlage zur Entscheidung der richtigen Auflösung zu gewährleisten, ist eine der Optionen der Kandidatenauswahl, alle erzeugten Kandidaten zu nutzen, ohne dass das Unigramm-basierte Auflösungsverfahren eine Vorauswahl trifft. Konkret bedeutet das, dass alle Wörter im Kontext eines Quelltextbezeichners, welche den gleichen Anfangsbuchstaben haben und alle Buchstaben in der gleichen Reihenfolge enthalten, wie ein betrachtetes Token einer jeden möglichen Spaltung, als Auflösungskandidat für dieses Token gelten. Hier wird der eigentliche Ansatz von Alatawi et al. zur Auflösung von Abkürzungen also nicht genutzt, sondern nur sein initialer naiver Schritt zur Kandidaten-Erzeugung. Es handelt sich bei der Entscheidung der Auflösung also um die isolierte Auswahl durch die worteinbettungsbasierte Ähnlichkeitsfunktion bezüglich der Token aus dem Kontext.

Eine zweite Option ist, alle gleichwertig besten Kandidaten nach vollendeter Auflösung durch den Alatawi et al. Unigramm-Ansatz zu verwenden. Einige Bezeichner erhalten nach dem Auflösungsschritt von Alatawi et al. genau eine als richtig erachtete Auflösung, bei anderen gibt es eine Mehrzahl gleichwertig bester Kandidaten, zwischen denen bislang willkürlich entschieden wird.

Während bei dieser Kandidatenauswahl vor der Nutzung des worteinbettungsbasierten Auflösungsschrittes die eindeutig besten offensichtlich genau so entschieden werden wie in der Imitation des Verfahrens von Alatawi et al., werden die mehrdeutig besten nicht mehr willkürlich, sondern mittels der worteinbettungsbasierten Ähnlichkeitsfunktion entschieden.

6.2.3 Ähnlichkeitsfunktionen

Im letzten Absatz von Abschnitt 5.4.4.3 wird bereits darauf eingegangen, dass die Bestimmung der Wahrscheinlichkeit eines Auflösungskandidaten eines abgekürzten Bezeichners in einem gewissen Kontext mittels Worteinbettungen einer gewissen Ähnlichkeitsfunktion bedarf, welche die Ähnlichkeit des Auflösungskandidaten zum Kontext, beziehungsweise der Wortvektoren bezüglich der Token aus Auflösungskandidaten und Kontext, modelliert. Für die paarweise Ähnlichkeit zweier Vektoren wird hier grundsätzlich die Cosinusähnlichkeit verwendet. Da der Kontext und der Auflösungskandidat, falls dieser zusammengesetzt ist, aber jeweils mehrere Vektoren enthalten, muss eine Funktion bestimmen, wie genau diese Vektoren miteinander verglichen werden.

6.2.3.1 Naiv-Brachial

Als naiver Ansatz zur Bestimmung der Ähnlichkeit zwischen einem gegebenenfalls zusammengesetzten Auflösungskandidaten und dem Kontext des Quelltextbezeichners werden alle Token des Auflösungskandidaten sowohl miteinander, als auch mit allen Token im Kontext verglichen und das Ergebnis aufsummiert und über die Menge an Summanden normalisiert. Dieser Ansatz kann gewissermaßen als „Brute-Force“ Strategie angesehen werden, da er rücksichtslos alle betrachteten Vektoren miteinander vergleicht.

6.2.3.2 Kontext-Brachial

Da der Ähnlichkeit der Token eines zusammengesetzten Auflösungskandidaten keine Aussagekraft zugemessen wird, wie in Abschnitt 5.4.4.3 begründet, gibt es eine ähnliche Ähnlichkeitsfunktion, welche nur die Ähnlichkeit der Token aus dem Auflösungskandidaten zu den Token aus dem Kontext berücksichtigt. Diese Ähnlichkeitsfunktion ähnelt stark der aus Abschnitt 6.2.3.1, nur dass sie auf die Vergleiche der Wortvektoren der Token aus dem Auflösungskandidaten untereinander verzichtet.

6.2.3.3 Kontext-Wortarten

Auf der Funktion aus Abschnitt 6.2.3.1 aufbauend nutzt eine weitere einen Wortartenmarkierer, um die einzelnen Token aus dem Auflösungskandidaten nur mit Token aus dem Kontext zu vergleichen, welche die gleiche Wortart darstellen. Um in dieser Variante der worteinbettungsbasierten Auflösungsverfahren die Wortarten der Auflösungskandidaten und der Token im Kontext zu berücksichtigen, wurde der Wortartenmarkierer der Stanford University *Stanford Log-linear Part-Of-Speech Tagger* verwendet [MSB+14]. Dieser weist Wörter verschiedenen Wortarten zu. Die Bibliothek erlaubt grundsätzlich die Verwendung verschiedener Markierer, hier wurde allerdings der Standard-Markierer *MaxentTagger* verwendet. Für die Zwecke dieser Arbeit werden die Wortarten vereinfachend den Gruppen Nomen, Verben, Adjektive beziehungsweise Adverbien und Sonstige zugewiesen.

6.2.3.4 Kontext-Aufgeteilt

Eine andere Ähnlichkeitsfunktion teilt die einzelnen Wörter des Kontextes gleichmäßig den Token des zusammengesetzten Auflösungskandidaten zu. Die Idee dahinter ist, dass jedes Token des zusammengesetzten Auflösungskandidaten mit einem anderen im Kontext beschriebenen Konzept zusammenhängt. Es wird also für jedes Token im Auflösungskandidaten eine Liste erzeugt, welche jeweils gleich viele Token aus dem Kontext enthält, deren Vektoren dem Vektor dieses Tokens aus dem Auflösungskandidaten am ähnlichsten sind. Die Zuteilung des jeweiligen Anteils des Kontextes zu einem Segment des zusammengesetzten Auflösungskandidaten findet gierig statt. Jedes Wort aus dem Kontext wird der Liste des jeweils diesem am nächsten liegenden Segmentes aus dem Auflösungskandidaten zugewiesen, es sei denn dass dessen Liste bereits die gewünschte Länge erreicht hat.

6.2.3.5 Kombinierte-Vektoren

Zuletzt gibt es eine Ähnlichkeitsfunktion, welche jeweils aus den Vektoren, die die Worteinbettungen der einzelnen Token des zusammengesetzten Auflösungskandidaten darstellen, und die Vektoren der Token aus dem Kontext, zu jeweils einem Vektor aufsummiert und um die Zahl der Summanden normalisiert, um so den Auflösungskandidaten und den Kontext jeweils als einen Vektor zu betrachten, und die Ähnlichkeit dieser beiden Vektoren zu bestimmen. Die Idee dieses Ansatzes ist, dass jeder dieser beiden Vektoren jeweils den gesamten Auflösungskandidaten, beziehungsweise den gesamten Kontext repräsentiert, und diese somit vollständig miteinander verglichen werden können.

6.3 Auswahl des betrachteten Kontextes

Für alle hier vorgestellten Abkürzungsauflösungsverfahren werden die Auflösungskandidaten aus einem gewissen Kontext im Quelltext um den abgekürzten Bezeichner herum gewonnen. Bei Alatawi et al. ist dieser Kontext immer die gesamte Quelltextdatei. In den hier vorgestellten Verfahren können verschiedene Kontexte ausgewählt werden, welche dann als Wissensquelle für die Suche nach Auflösungskandidaten genutzt werden.

Hierfür gibt es drei Parameter. Der erste entscheidet den grundlegenden betrachteten Kontext. Im Syntaxbaum wird von einem betrachteten Bezeichner aus so lange der nächst höhere Kontext ausgewählt, bis er entweder diesem im Parameter entschiedenen Kontext entspricht, oder einem noch höheren. Die so konfigurierbaren Kontexte sind die Anweisung, der Quelltextblock, die Methode, die Klasse oder das Paket, in welchem sich der betrachtete, potentiell abgekürzte Quelltextbezeichner befindet. Ist der gewünschte Kontext zum Beispiel die Methode, so wird vom Bezeichner aus beispielsweise zunächst die Anweisung, dann der Quelltextblock, und dann die Methode betrachtet. Hier endet die Suche, und der Kontext der Methode besteht aus allem darin enthaltenen Quelltext und allen Kommentaren, welche der Methode und ihrem Inhalt zugeschrieben werden. Wird vom Bezeichner aus bei der Suche nach dem gewünschten Kontext allerdings eine Klasse erreicht, bevor eine Methode erreicht wird, wird dort abgebrochen und der Kontext der Klasse verwendet. Kontexte, welche noch kleiner sind als die Methode, werden allerdings nicht für erstrebenswert für die Kandidatensuche erachtet, da eine große Gefahr besteht, dass diese den richtigen Auflösungskandidaten nicht enthalten.

Der zweite und dritte Parameter zur Auswahl des gewünschten Kontextes entscheiden, ob Methodenaufrufe vor- beziehungsweise zurückverfolgt werden. Hierzu muss der im ersten Parameter ausgewählte Kontext die Methode, die Klasse oder das Paket sein, da sonst keine Methodenaufrufe innerhalb des Kontextes gefunden werden. Ist dies der Fall kann, mit den entsprechenden Parametern, der Kontext um den Kontext der Methoden erweitert werden, welche im durch den ersten Parameter beschlossenen Kontext aufgerufen werden, oder welche einen Aufruf der Methoden enthalten, die im durch den ersten Parameter beschlossenen Kontext definiert werden.

6.4 Wörterbücher

Um zu überprüfen, ob es sich bei einem Quelltextbezeichner, einem Token eines Quelltextbezeichners, oder einer Abkürzungsauflösung um ein natürlichsprachliches Wort handelt, kann ein Wörterbuch eingesetzt werden, um Quelltextbezeichner, welche darin gefunden werden, nicht aufzulösen, sondern davon auszugehen, dass diese unverändert bleiben sollten. Dies wird als optionaler Schritt vor der Auflösung eines Quelltextbezeichners ausgeführt. Dieser Schritt birgt das Potential zu verhindern, dass nicht natürlichsprachliche Quelltextbezeichner fälschlicherweise aufgelöst werden. Allerdings könnte er auch dazu

führen, dass abgekürzte Bezeichner fälschlicherweise als Wörterbuch-Wörter erkannt, und deshalb nicht aufgelöst werden, weshalb dieser Schritt optional ist.

Um bestimmte Wörter nicht als Auflösungskandidaten zu berücksichtigen, die als besonders unwahrscheinlich und potentiell störend befunden werden, wird eine bereits von Alatawi et al. [AXX17] [AXY18] verwendete Stoppwortliste zu Rate gezogen. Diese dient dazu zu verhindern, dass Stoppwörter wie „the“, „that“, „this“ oder „etc“, welche eventuell häufig im Quelltext oder in den Kommentaren auftreten, aber unwahrscheinlich die richtige Auflösung einer Abkürzung darstellen, als Auflösungskandidaten für Abkürzungen in Betracht gezogen werden.

7 Implementierung

In Kapitel 3 wurde die Projektumgebung beschrieben, welche für die Umsetzung aller hier vorgestellten Verfahren verwendet wurde. Für alle Verfahren wird zunächst der Graph, welcher das Absichtsmodell auf Quelltextseite darstellt, mit dem Quelltext des betrachteten Beispielquelltextes befüllt. Anschließend wird dieser Graph für die Abkürzungsauflösung vorbereitet, indem er um Knoten erweitert wird, welche die Quelltextbezeichner darstellen, und um Attribute, welche den Kontext, der für die Suche nach Auflösungskandidaten genutzt wird, darstellen. Wie diese Kontexte genutzt werden wird in Abschnitt 6.3 beschrieben. Anschließend wird ein Agent ausgeführt, welcher ein bestimmtes Abkürzungsauflösungsverfahren umsetzt. Dieser Agent liest und schreibt auf dem Graphen. Zuletzt wird ein Agent ausgeführt, welcher die Ergebnisse anhand der entsprechenden Musterlösung des betrachteten Beispielquelltextes evaluiert. Dieser Ablauf wird in Abbildung 7.1 illustriert.

7.1 Einlesen des betrachteten Quelltextes

Um das Absichtsmodell bezüglich des Quelltextes aufzubauen, welches in Kapitel 3 eingeführt wurde, muss der betrachtete Beispielquelltext in den Graphen eingelesen werden. *INDIRECT* beinhaltet hierzu einen Agenten [Eur20] für Java-Projekte, welcher auf einen Antlr-Parser [Par21] nutzt, um einen abstrakten Syntaxbaum zu erzeugen. Dieser abstrakte Syntaxbaum kann anschließend von weiteren Agenten verändert und angereichert werden, um das Absichtsmodell auf Quelltextseite zu erzeugen. Allerdings setzt dies ein Java-Projekt als Beispielquelltext voraus. Dieser in *INDIRECT* inbegriffene Agent zur Erzeugung des Absichtsmodells kann von den für die Evaluation betrachteten Beispielquelltexten also nur für Java-Programme eingesetzt werden. Programme, welche in anderen Programmiersprachen geschrieben wurden, oder unzusammenhängende Quelltextbeispiele, kann dieser Agent allerdings nicht eingesetzt werden. Um mit Beispielquelltexten arbeiten zu können, welche keine Java-Projekte sind, wurden zwei weitere Agenten implementiert. Einer ist ein Pseudo-C-Parser, welcher ebenfalls auf Antlr basiert, und einen Graphen aufbaut, welcher einem abstrakten Syntaxbaum ähnlich genug ist, um die verschiedenen in Abschnitt 6.3 vorgestellten möglichen Kontexte zur Kandidatensuche zu ermöglichen, nicht aber die Rückverfolgung von Methodenaufrufen. Der zweite Agent, welcher zur Absichtsmodell-Erzeugung implementiert wurde, liest schlicht den Inhalt jeder Quelltextdatei in je einen Knoten des Graphen, welcher dazu dient eine Klasse zu repräsentieren. Hierbei werden Kommentare, die anhand der Syntax von Java beziehungsweise C

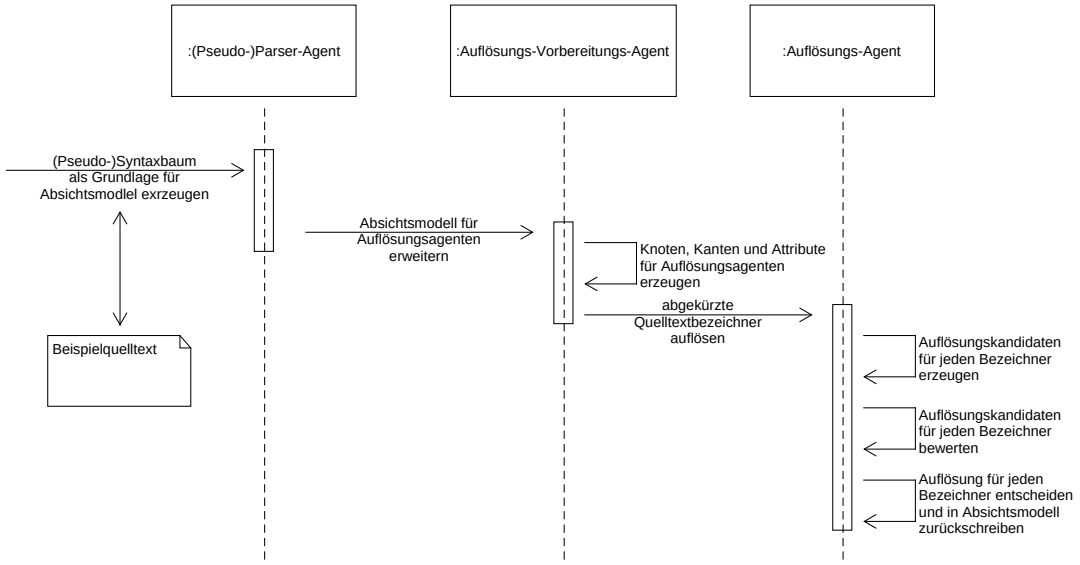

Abbildung 7.1: Sequenzdiagramm der Abkürzungsauflösung

erkannt werden, in das entsprechende Attribut des Knotens gespeichert. Der restliche Text wird als Quelltext angesehen. Bei Verwendung dieses Absichtsmodell-Erzeugungs-Agenten ist es also nicht möglich, tiefere Kontexte als die ganze Quelltextdatei, welche als Klasse interpretiert wird, zu betrachten.

In jedem Fall wird der Graph anschließen, durch einen eigens dafür implementierten Agenten, um weitere Informationen angereichert, um die Abkürzungsauflösung vorzubereiten. Es werden ein neuer Knotentyp und ein neuer Kantentyp für Quelltextbezeichner angelegt. Verschiedene Knoten im Baum, welche bestimmte Anteile des Quelltextes wie Anweisungen, Quelltextblöcke, Methoden oder Klassen darstellen, werden mit den Knoten verknüpft, welche die Bezeichner darstellen, und erhalten ein neues Attribut, welches den Kontext dieses Elementes darstellt. Der Knoten des Bezeichners wiederum wird mit der Quelltextzeile und dem Namen der Quelltextdatei versehen, in welchen er auftritt. Außerdem erhält der Knoten des Bezeichners ein Attribut, in welches der Auflösungsagent die Auflösung zurückschreibt. Der Kontext, welcher bestimmten Knoten im Graphen zugewiesen wird, wird bei den verschiedenen Auflösungsverfahren genutzt, um darin nach Auflösungskandidaten zu suchen. Wird beispielsweise die Klasse als der zur Kandidatensuche verwendete Kontext ausgewählt, dann werden alle Bezeichner innerhalb einer Klasse aufgelöst, indem der Quelltext und die Kommentare innerhalb der Klasse nach Auflösungskandidaten durchsucht werden, und zwischen diesen Kandidaten mittels des jeweiligen Auflösungsverfahrens entscheiden wird. Eben dieser Kontext wurde zuvor im jeweiligen Knoten des Graphen abgelegt.

7.2 Implementierung der Ansätze von Alatawi et al.

Der Quelltext zu den Verfahren von Alatawi et al. [AXX17] [AXY18] ist, in Python implementiert, öffentlich zugänglich. Um die hier vorgestellten Verfahren wie gewünscht im Rahmen des in Java implementierten *INDIRECT*-Projektes [Hey19] umsetzen zu können, wurde der Quelltext manuell in Java übersetzt, beziehungsweise neu implementiert. Sowohl der Wechsel der Programmiersprache, als auch der Einsatz des in *INDIRECT* bereitgestellten Syntaxbaumes, führten gezwungenermaßen zu Änderungen in der Struktur des Quelltextes. Hinzu kommen einige Abweichungen in der Architektur, welche die

Implementierung modularer gestalten, um Vererbung und Erweiterung zu ermöglichen, welche für die Umsetzung der hier vorgestellten Innovationen nötig sind. Die Ergebnisse dieser neuen Implementierung, bei der Evaluation anhand der gleichen Beispiele, sind nicht ganz identisch mit denen des Originals. Offenbar ist es nicht gelungen das Verfahren vollwertig zu imitieren. Es wurde aber kein semantischer Unterschied zwischen den beiden Implementierungen gefunden, sodass die Imitation nicht korrigiert werden konnte. Die Abweichungen sind allerdings nicht allzu groß, und die Ergebnisse der neuen Implementierung auf den originalen Beispielen können dennoch als Richtwert angesehen werden, um relative Unterschiede zu den neuen Verfahren zu erkennen.

In der Implementierung von Alatawi et al. [AXY18] existiert ein Fehler, der dazu führt, dass sie im Bigramm-basierten Verfahren die selbe Unigramm-Grammatik für die Suche nach Bigrammen im Quelltext nutzen, wie für Unigramme. Die Wahrscheinlichkeit der Auflösungskandidaten wird anhand einer Uni- und einer Bigramm-Grammatik ermittelt, wie in Abschnitt 5.4.4.1 beschrieben. Die Wahrscheinlichkeit der Quelltextbezeichner wird zwar mittels der gleichen Formel berechnet, beim Auslesen der Wahrscheinlichkeit aus der Statistik wird aber auf die Unigramm-Grammatik zugegriffen, in der offensichtlich keine Bigramme zu finden sind. Debugging des originalen Python-Quelltextes hat ergeben, dass hierbei tatsächlich kein einziges der Bigramme in der Quelltext-Unigramm-Grammatik gefunden wird. Um die Vergleichbarkeit mit dem Original zu gewährleisten, aber gleichzeitig diesen Fehler vermeiden zu können, wurde eine Konfiguration implementiert, welche diesen Fehler wahlweise imitiert, oder während der Ausführung des Auflösungsverfahrens eine neue Unigramm-, Bigramm- und Trigramm-Grammatik auf dem Quelltext erzeugt. Dieses Feature hat außerdem den Vorteil, dass es für andere Beispielprojekte passende N-Gramm-Grammatiken für den Quelltext erzeugt, welche nicht vorgefertigt zur Verfügung stehen.

Ein weiterer Unterschied ist, dass in der originalen Implementierung von Alatawi et al. die aufzulösenden Abkürzungen aus der Musterlösung ausgelesen, und nur die Auflösungskandidaten im Quelltext gesucht werden. Dies steht zum einen im Konflikt zum Ziel dieser Arbeit, Abkürzungen in Quelltext zu erkennen und aufzulösen, und zum anderen mit der angestrebten Definition von Erfolg. Denn es soll in der Evaluation dazwischen unterschieden werden, ob ein zusammengesetzter Bezeichner vollständig richtig aufgelöst wurde, oder einzelne Abkürzungen daraus. Zwei der Abkürzungen in der Musterlösung von Alatawi et al. treten allerdings nur als Teil eines anderen Bezeichners auf, während andere eigenständige Bezeichner sind. In der Imitation und den darauf aufbauenden Verfahren werden die gegebenenfalls aufzulösenden Bezeichner also aus dem Quelltext gewonnen.

Zudem lässt die originale Implementierung von Alatawi et al. beim Verfahren für Bigramm-basierte Auflösung keine Auflösungskandidaten zu, welche nicht aus mindestens zwei Token zusammengesetzt sind. Dies wird in Abschnitt 5.4.4.1 erörtert und begründet und in Abschnitt 6.1 auf das Trigramm-basierte Verfahren erweitert. Dies ist zwar im Bezug auf die Musterlösung von Alatawi et al. legitim, da dort tatsächlich alle Abkürzungen aus mindestens zwei Token zusammengesetzt sind, lässt sich aber nicht auf alle Anwendungsfälle verallgemeinern. Deshalb werden in der imitierten Implementierung Quelltextbezeichner, für welche nur Auflösungskandidaten in Frage kommen, welche aus nur einem Token bestehen, mittels des Unigramm-basierten Verfahrens aufgelöst. Dies macht im Bezug auf die Musterlösung von Alatawi et al. keinen Unterschied, da dort nur abgekürzte Quelltextbezeichner vorkommen, welche aus mindestens zwei Token zusammengesetzt sind. Somit werden auch in der imitierten Implementierung Auflösungskandidaten, welche aus nur einem Token bestehen, genau wie im Original, nie berücksichtigt, da es immer eine Alternative mit mehr als einem Token gibt. Für Bezeichner aus anderen Beispielen, welche aus nur einer Abkürzung bestehen, liefert das Verfahren so aber auch Ergebnisse, während die originale Implementierung von Alatawi et al. diese vollständig ignorieren würde. Es bleibt

aber der Nachteil, dass, sobald ein Auflösungskandidat, welcher aus mehr als einem Token besteht, in Frage kommt, der vielleicht richtige Auflösungskandidat, welcher aus nur einem Token besteht, gar nicht erst berücksichtigt wird.

7.3 Worteinbettungsverfahren

Um *word2vec*-Modelle zu laden und auf diese zuzugreifen, wird „Java Native Access“ [jna21] verwendet, um auf ein Golang-Programm zuzugreifen, welches mittels des *word2vec*-Pakets von Sajari [OHI21] die nötigen Methoden für die Verwendung des *word2vec*-Modells bereitstellt. Diese Schnittstelle wird verwendet um die Wort-Vektoren der betrachteten Token zu laden, welche dann entsprechend einer der Ähnlichkeitsfunktionen aus Abschnitt 6.2.3 verarbeitet werden. *fastText*-Modelle werden mittels eines in Python implementierten Servers geladen und bereitgestellt, welcher als Teil des *INDIRECT* Projektes in der Masterarbeit „Entwurf und Aufbau einer semantischen Repräsentation von Quelltext“ von Eurich [Eur20] implementiert wurde, und hier eine Schnittstelle zum Laden der Wortvektoren bietet, analog zu der für *word2vec*-Modelle.

8 Datensätze

Für die Umsetzung der in Kapitel 6 vorgestellten Verfahren wurden verschiedene Datensätze eingesetzt, die als Wissensquelle für die Auswahl des besten Auflösungskandidaten, für die Filterung von Auflösungskandidaten oder aber als Beispielquelltext oder Musterlösung für die Evaluation der verschiedenen Herangehensweisen dienen. In diesem Kapitel werden diese Datensätze, ihre Ursprünge und Einsatzzwecke dargelegt.

8.1 Beispielprojekte

Alatawi et al. stellen, zusammen mit dem Python Quelltext ihrer beiden Verfahren [AXX17] [AXY18], einige beispielhafte Java-Quelltextdateien und eine Musterlösung zur Verfügung. Die Musterlösung enthält 100 Abkürzungen, für welche es jeweils eine Beispieldatei geben soll. Allerdings befinden sich in der frei zugänglichen Quelle nur 96 dieser Dateien. Zudem handelt es sich bei den Dateien um unzusammenhängende und nicht abgeschlossene Quelltextausschnitte. Dies ist für die Verfahren von Alatawi et al. kein Problem, da sie grundsätzlich von der Quelltextdatei, in welcher eine Abkürzung gefunden wird, als Quelle ausgehen. Dieser Umstand beschränkt die Evaluation der hier vorgestellten Verfahren darauf das gleiche zu tun, da kein stimmiger Syntaxbaum über diesen nicht zusammenhängenden Quelltext aufgebaut werden kann, und die jeweils um eine bestimmte Abkürzung herum zurechtgestutzten, unzusammenhängenden Kontexte willkürlich aneinander zu reihen offensichtlich weder realitätsnah noch zielführend wäre. Außerdem muss, zum Einlesen dieses Beispielquelltextes in den Absichtsmodell-Graphen, der in Abschnitt 7.1 vorgestellte naive Pseudo-Parser eingesetzt werden, da aus den unzusammenhängenden Quelltextausschnitten kein echter Syntaxbaum erzeugt werden kann.

Um die Vergleichbarkeit mit verwandten Arbeiten gewährleisten zu können, und umfangreichere reale Software-Projekte als Beispiele für die Evaluation heranzuziehen, wurden die von Lawrie et al. zur Evaluation ihres *Normalize*-Ansatzes [LB11] verwendeten Projekte *a2ps* und *which* als Beispiele herangezogen. Diese wurden auch von Coraza et al. zur Evaluation ihres *LINSEN*-Ansatzes [CDMM12] verwendet. Beide Projekte wurden in C implementiert. *a2ps* hat insgesamt über 52.672 Quelltextzeilen, *which* hat 3.280 Quelltextzeilen. Um diese Beispielprojekte in den Absichtsmodell-Graphen einzulesen, muss der in Abschnitt 7.1 vorgestellte Pseudo-C-Parser eingesetzt werden.

Hinzu wurde der frei verfügbare Quelltext des Steuerungssystems für unbemannte Flugobjekte *Dronology* [ND21] als Beispiel herangezogen. *Dronology* befindet sich derzeit in

Version 0.07, ist also nicht abgeschlossen. Es bietet allerdings gerade deswegen den Vorteil, dass es, angesichts des noch geringen Umfanges, in überschaubarer Zeit möglich war, eine Musterlösung für den Quelltext des Projektes herzustellen. Außerdem ist die Software eindeutig einer Domäne zuzuordnen, nämlich der Steuerung von Drohnen beziehungsweise unbemannten Flugobjekten, sodass unschwer entschieden werden kann, welche Wissensquellen, wie Domänenliteratur oder Artfakte aus weiteren Projekten aus der gleichen Domäne, zur Abkürzungsauflösung hinzu gezogen werden sollte.

Außerdem wurde ein einfaches „Hello World“ Projekt geschrieben, welches eine Liste von strukturellen Besonderheiten enthält, die für die Abkürzungsauflösung interessant sein könnten. Dieses beinhaltet die in Abschnitt 6.3 bereits erwähnten Kontexte, aus welchen Auflösungskandidaten, je nach Konfiguration, gewonnen werden können. Darüber hinaus beinhaltet das „Hello World“ Projekt Schnittstellen, welche implementiert werden, und vererbte Klassen. Diese Vererbungen werden bislang beim Aufbau des Kontextes nicht berücksichtigt, das „Hello World“ Projekt hält sie aber für die Nutzung in zukünftigen Arbeiten bereit. In dem Projekt wurden bewusst sowohl Präfix-Abkürzungen, als auch Abkürzungen mit weggelassenen Buchstaben, Akronyme und Bezeichner, welche aus mehreren Abkürzungen zusammengesetzt sind, benutzt. Einige der Abkürzungen aus der Musterlösung sind in Beispiel 8.1 zu sehen.

Beispiel 8.1: Abkürzungen aus dem „Hello World“ Projekt

- „str“ als Präfix-Abkürzung für „string“
- „cntnt“ als Abkürzung mit weggelassenen Buchstaben für „content“
- „cp“ als Akronym für „child printer“
- „nst“ als kombinierte Multiwort-Abkürzung für „new string“

8.2 Musterlösungen

Die Musterlösung, welche Alatawi et al. für ihren Beispielquelltext zu Verfügung stellen, beinhaltet die Quelltextdateien und Zeilen, in welchen ein abgekürzter Quelltextbezeichner auftritt. Somit werden in dieser Musterlösung bestimmte Vorkommnisse von eventuell wiederkehrenden Quelltextbezeichnern gemessen. Verfahren, welche mit dieser Musterlösung evaluiert werden, lassen sich also bezüglich des konkreten Auftretens eines Bezeichners in einem bestimmten Kontext evaluieren, anstatt mehrfach auftretende Bezeichner mehrfach aufzulösen, sie aber in der Musterlösung jeweils auf die gleiche Auflösung abzubilden. Gleichzeitig wird fast jede Abkürzung in dieser Musterlösung aber auch nur einmalig erwähnt, nur elf werden zweimal in verschiedenen Kontexten erwähnt. Ob eine Abkürzung in einem anderen Kontext, in welchem sie eventuell noch einmal auftritt, wieder richtig aufgelöst würde, bleibt offen. Es ist fragwürdig, ob eine mehrfach auftretende Abkürzung mehrfach richtig oder mehrfach falsch aufzulösen nur einmal gewertet werden sollte. Wenn eine Abkürzung mal richtig und mal falsch aufgelöst wird, ist dies durchaus relevant für die Güte des Verfahrens. Aber auch mehrfach richtig oder mehrfach falsch aufgelöste Abkürzungen sind für den Versuch, dem Quelltext Informationen zu entnehmen, nicht unbedeutend. Außerdem ist bezüglich der Musterlösung von Alatawi et al. hervorzuheben, dass sie sich auf kein vollständiges, reales Software-Projekt bezieht, sondern auf eine Vorauswahl an 96 beispielhaften Quelltextausschnitten, welche jeweils für eine bestimmte Abkürzung als Kontext vorbestimmt sind. Dies könnte die externe Validität der Ergebnisse der Evaluation bezüglich dieser Musterlösung schmälern. Tatsächlich treten von diesen 96 abgekürzten Bezeichnern aus der Musterlösung fünf nicht als solche im Quelltext der jewei-

ligen Beispieldatei auf. Zwei treten nur als Abkürzung innerhalb eines anderen Bezeichners auf, drei treten gar nicht im entsprechenden Quelltext auf. Die Liste ist im Anhang in Abschnitt B.1 zu finden. Daher können die hier vorgestellten Verfahren, welche die potentiell aufzulösenden Bezeichner im Quelltext suchen, nur anhand der übrigen 91 Einträge von 80 verschiedenen abgekürzten Bezeichnern evaluiert werden.

Die Musterlösungen, welche durch Lawrie et al. [LB11] für die Programme *a2ps* und *which* zur Verfügung gestellt wurden, enthalten hingegen keine Hinweise auf den Ort des Auftretens eines abgekürzten Bezeichners. Somit werden hier die Bezeichner gegebenenfalls mehrfach im Quelltext gefunden, und ihre Auflösung, ob richtig oder falsch, mehrfach mit der in der Musterlösung verglichen. Das Urteil über den Erfolg des Verfahrens bezieht sich hier also mehr auf alle Vorkommnisse aller abgekürzten Bezeichner, während ein Verfahren auf der Musterlösung von Alatawi et al. bezüglich bestimmter einmaliger Vorkommnisse abgekürzter Bezeichner evaluiert wird. Auch in den Musterlösungen von *a2ps* und *which* treten Abkürzungen auf, welche nicht im Quelltext zu finden sind. Von den 205 Einträgen der Musterlösung von *a2ps* werden 177 insgesamt 1.846 mal gefunden. Die Musterlösung für Abkürzungen in *which* enthält 474 Einträge, davon werden 299 insgesamt 2.242 mal gefunden. Dass diese mehrfach gefunden werden, liegt an der Natur dieser Musterlösungen, anhand derer die gleiche Abkürzung mehrfach ausgewertet werden kann. Diese Zahlen sind für alle Abkürzungsauflösungsverfahren gleich, da die Suche von potentiell aufzulösenden Bezeichnern im Quelltext für alle hier vorgestellten Verfahren die gleiche ist.

Für die hier vorgestellte Arbeit wurden eigens drei Musterlösungen für den Quelltext von *Dronology* angelegt. Erstere, im Stil der Musterlösungen von Lawrie et al., enthält alle 120 abgekürzten Bezeichner aus *Dronology*, allerdings ohne Hinweis auf Quelltextdatei und Quelltextzeile. Die zweite, im Stil der Musterlösung von Alatawi et al., beschränkt sich auf vier willkürlich ausgewählte Quelltextdateien aus dem 44 Dateien umfassenden Quelltext von *Dronology*, beinhaltet allerdings alle darin auftretenden, abgekürzten Bezeichner mit Hinweis auf die Quelltextdatei und Quelltextzeile. Im Gegensatz zu der Musterlösung von Alatawi et al. enthält diese Musterlösung mehrfach auftretende Bezeichner tatsächlich mehrfach, mit Verweis auf die einzelnen Orte, an welchen diese auftreten. Diese Musterlösung enthält 77 Vorkommnisse von 22 verschiedenen abgekürzten Quelltextbezeichnern. Diese sind in Abschnitt B.2 zu finden. Eine dritte Musterlösung, auf den gleichen vier Quelltextdateien, beinhaltet ebenfalls Informationen über die Quelltextdatei und die Quelltextzeile, in welcher ein Bezeichner auftritt, und zwar nicht nur für alle abgekürzten, sondern für alle Bezeichner in den vier Quelltextdateien. Der Zweck dieser Musterlösung ist zu evaluieren, wie viele nicht abgekürzte Bezeichner durch verschiedene Verfahren, beziehungsweise in verschiedenen Konfigurationen der Verfahren, fälschlicherweise als Abkürzungen betrachtet und aufgelöst werden. Diese Musterlösung enthält 513 Einträge. Auch bei diesen drei Musterlösungen werden nicht alle eingetragenen Bezeichner gefunden, da einige Token aus den Kommentaren eingetragen wurden. Genau werden aus der ersten Musterlösung 113, aus der zweiten alle 77, und aus der dritten 498 der Einträge im Quelltext gefunden.

Auch für das „HelloWorld“ Projekt wurde eine Musterlösung angelegt, welche, im Stil der Musterlösungen von Lawrie et al., jede Abkürzung nur einmal enthält, ohne Verweis auf Quelltextdatei oder Zeile. Dieses Beispielprojekt mit insgesamt acht verschiedenen Abkürzungen ist allerdings, da es explizit als Beispielprojekt für die Auflösung von Abkürzungen erzeugt wurde, wenig aussagekräftig, auch weil Hinweise für die Auflösungen der Abkürzungen bewusst platziert wurden. Es dient deshalb nicht der Evaluation der Verfahren an sich, sondern nur zum Testen des erwartungsgemäßen Verhaltens der Software.

8.3 Wörterbücher

Als Wörterbuch für den optionalen Schritt des Überspringens von Wörterbuchwörtern bei der Auflösung, welcher in Abschnitt 6.4 vorgestellt wurde, wird das „Webster's Unabridged Dictionary“ [Web21] zu Rate gezogen, welches in Form einer JSON Datei geladen wurde.

8.4 N-Gramm-Grammatiken

Wie bereits in Abschnitt 5.4.3 erörtert wurde, stellen Alatawi et al. die Software-basierter Uni- und Bigramm-Grammatiken [XXA+18] und die natürlichsprachlichen Uni- und Bigramm-Grammatiken [BF06], welche sie für die Evaluation ihrer eigenen Verfahren verwendet haben, online zur Verfügung. Die aus Software-basierten Quellen gewonnenen N-Gramm-Grammatiken tragen den Namen „Source Code Repository“, kurz „SCR“, und die Quelle aus der sie gewonnen wurden sind 0,7 Millionen Software-Projekte, die auf Github [Git21] veröffentlicht wurden. Die aus natürlicher Sprache stammenden N-Gramm-Grammatiken, unter dem Namen „Natural Language Repository“ oder „NLR“, wurden aus einem Terabyte online verfügbaren Textdaten [BF06], wie Bücher und Nachrichtenartikel, gewonnen.

Die für den Trigramm-basierten Ansatz benötigten Trigramm-Grammatiken wurden dem *Google Books Ngram Viewer* Korpus „English One Million“ [Goo21b] entnommen. Um die Uni- Bi- und Trigramm-basierten Verfahren unabhängig von der Wissensquelle, aus welcher die N-Gramme stammen, gegeneinander evaluieren zu können, wurden dem *Google Books* N-Gramm-Korpus außerdem Uni- und Bigramm-Grammatiken entnommen. Da der *Google Books* N-Gramm-Korpus Informationen enthält, die hier nicht berücksichtigt werden, wie die Zahl der Bücher in welchen ein N-Gramm vorkommt, oder die Zahl der Auftritte eines N-Gramms in Büchern welche in einem gewissen Jahr erschienen sind, wurden diese Daten für die Verwendung in den N-Gramm-basierten Verfahren vereinfacht. Für jedes N-Gramm wird nur die Gesamtzahl der Vorkommnisse in allen Büchern im Korpus betrachtet, und alle N-Gramme, welche Wörter aus der Stoppwortliste des „Natural Langugae Toolkit“ [NLT21] oder aus der Stoppwortliste von Alatawi et al. [AXY18] enthalten, wurden entfernt, da die Abkürzungen nicht zu Stoppwörtern aufgelöst werden sollten, und das Verfahren verbietet sie zu den Wörtern aus der Stoppwortliste von Alatawi et al. aufzulösen.

All diese N-Gramm-Grammatiken dienen der Bestimmung der Auftretenswahrscheinlichkeiten der Auflösungskandidaten. Für die Auftretenswahrscheinlichkeiten der Abkürzungen stellen Alatawi et al. eine aus ihrem eigenen Beispielquelltext gewonnene Unigramm-Grammatik zur Verfügung. Außerdem können die N-Gramme für das jeweilige Verfahren während der Ausführung des Auflösungsverfahrens direkt aus dem betrachteten Quelltext gewonnen werden.

8.5 Worteinbettungsmodelle

Wie in Abschnitt 5.4.4.3 begründet und in Abschnitt 6.2.1 dargelegt wird, wurden sowohl für die Verwendung von *word2vec*, als auch die von *fastText*, sowohl im Voraus trainierte, als auch explizit für diese Arbeit trainierte Worteinbettungsmodelle eingesetzt.

Das im Voraus trainierte *word2vec*-Modell, welches hier verwendet wurde, ist das „Word Embeddings for the Software Engineering Domain“ Modell [ECS18], welches auf Basis von Beiträgen auf *Stack Overflow* [Sta21] trainiert wurde, und somit einen Software-bezogenen Datensatz darstellt. Das Software-basierte *word2vec*-Worteinbettungsmodell wurde gewählt, da ein Software-basiertes Modell vermutlich bessere Ergebnisse auf einem Software-bezogenen Verfahren erzielt.

Als *fastText*-Modell wurde hingegen ein auf *Common Crawl* [Cra21] trainiertes und von Facebook bereitgestelltes Modell [Fac21] verwendet, welches dementsprechend ein natürlichsprachlicher Datensatz ist. Vermutet wird, dass Software-basierte Datensätze für die Auflösung von Abkürzungen in Quelltext die bessere Wissensquelle darstellen. Ein solches *fastText*-Modell wurde leider nicht gefunden. Allerdings könnten die Subwortinformationen, welche *fastText* verwendet, einen Vorteil darstellen, wie in Abschnitt 5.4.4.3 erörtert wurde. Das *fastText*-Modell wurde gewählt, da es in Form von „Common Crawl“ einen breiten natürlichsprachlichen Korpus abdeckt, und somit für das beste verfügbare Modell befunden wurde.

Zusätzlich wurden Worteinbettungsmodelle explizit für die Evaluation dieser Arbeit erzeugt. Da diese sowohl für *word2vec* als auch für *fastText* auf den gleichen Wissensquellen basieren, können diese auch verwendet werden, um einen direkten Vergleich der Vor- und Nachteile der Worteinbettungsverfahren für diesen Verwendungszweck zu erhalten, welcher nicht durch die Wahl des Modells verfälscht wird. Für alle Beispielprojekte, welche bei der Evaluation verwendet werden, wurde je ein *fastText*- und ein *word2vec*-Modell auf dem Quelltext selbst erzeugt. Zudem wurde für die Evaluation der worteinbettungsbasierten Verfahren auf dem Quelltext von *Dronology* [ND21] je ein *fastText*- und ein *word2vec*-Worteinbettungsmodell erzeugt, welches als Wissensquelle eine Sammlung von acht frei zugänglichen Programmen zur Steuerung von Drohnen, darunter *Dronology* selbst, verwendet. Eine Liste der Programme ist in Tabelle A.1 zu finden. In Abschnitt 5.4.4.3 wird darauf eingegangen, dass ein zu großer Korpus dazu führen könnte, dass Wörter als naheliegende Auflösungskandidaten befunden werden könnten, welche in der betrachteten Domäne nicht relevant sind. Umgekehrt könnte es sein, dass ein zu kleiner Korpus die richtige Auflösung nicht enthält. Von einer Auswahl mehrerer Projekte aus der gleichen Domäne als Wissensquelle für das Training eines Worteinbettungsmodells wird also erwartet, dass diese beiden Probleme vermieden werden. Zuletzt wurde je ein *fastText*- und ein *word2vec*-Modell auf dem englischen Wikipedia-Artikel über unbemannte Flugobjekte [Wik21b] trainiert. Dieses Modell soll als Beispiel für nicht Software-bezogene, aber Domänen-bezogene Wissensquellen dienen. Alle diese selbst trainierten Modelle wurden mittels *gensim* [eL09] mit der Standardvektorlänge von 100 erzeugt. Angesichts der verhältnismäßig kleinen verwendeten Wissensquellen wurde es nicht für nötig befunden, größere Vektoren zu verwenden, und umgekehrt bestand auch kein Bedarf, zu Gunsten der Berechnungseffizienz kleine Vektoren zu verwenden.

9 Evaluation

In diesem Kapitel werden die in dieser Arbeit vorgeschlagenen Verfahren zur Auflösung von Abkürzungen in Quelltext evaluiert. Die hier eingeführten Innovationen werden anhand folgender Forschungsfragen bezüglich ihrer Tauglichkeit für die Auflösung von abgekürzten Quelltextbezeichnern evaluiert:

- **Forschungsfrage 1:** Wie groß ist der Einfluss des Auflösens, beziehungsweise Nichtauflösens von Wörterbuch-Wörtern auf die Präzision der Verfahren von Alatawi et al.?

Wie in Abschnitt 6.4 diskutiert wird, kann das Nichtauflösen von Wörterbuch-Wörtern einerseits die Zahl falsch-positiver Auflösungen, also Auflösungen von Bezeichnern, welche keiner Auflösung bedürfen, verringern, zum anderen kann es aber auch die Zahl falsch-negativer erhöhen, indem Abkürzungen, welche homonym mit Wörterbuch-Wörtern sind, aufgrund dieses Wörterbucheintrages ignoriert werden. **Forschungsfrage 1** dient dazu, diese Vor- und Nachteile zu quantifizieren, um abwägen zu können, ob oder wann die Vorteile es wert sind, die Nachteile in Kauf zu nehmen.

- **Forschungsfrage 2:** Wie hoch ist die Präzision des Trigramm-basierten Verfahrens im Vergleich zu den Verfahren von Alatawi et al.?

Mit der Beantwortung von **Forschungsfrage 2** soll überprüft werden, ob das Weiterführen des Ansatzes von Alatawi et al. mit größeren N-Grammen, wie in Abschnitt 5.4.4.1 vorgeschlagen und in Abschnitt 6.1 konkretisiert, das Potential hat, bessere Ergebnisse bei der Abkürzungsauflösung zu erzielen.

- **Forschungsfrage 3:** Welche der vorgeschlagenen Ähnlichkeitsfunktionen für die Abkürzungsauflösung auf Basis von Worteinbettungen liefert die größte Präzision?

In Abschnitt 5.4.4.3 wurde dargelegt, dass die Auswahl eines Auflösungskandidaten mittels Worteinbettungen einer Ähnlichkeitsfunktion bedarf, welche zur Einschätzung der Wahrscheinlichkeit des Kandidaten im Kontext des abgekürzten Quelltextbezeichners dient. In Abschnitt 6.2.3 wurden einige solche Ähnlichkeitsfunktionen vorgeschlagen. Um die hier vorgestellte Abkürzungsauflösung auf Basis von Worteinbettungen unter den bestmöglichen Bedingungen zu evaluieren, muss **Forschungsfrage 3** beantwortet werden, sodass die beste verfügbare Ähnlichkeitsfunktion bei der Evaluation des Verfahrens eingesetzt wird.

- **Forschungsfrage 4:** Welches Worteinbettungsverfahren liefert als Grundlage für die hier vorgestellten worteinbettungsbasierten Abkürzungsauflösungsverfahren die höhere Präzision: *fastText* oder *word2vec*?

In Abschnitt 5.4.4.3 wurden einige Worteinbettungsverfahren vorgestellt, und anschließend in Abschnitt 6.2 begründet, warum *fastText* und *word2vec* als die vielversprechendsten unter ihnen angesehen werden. Um, ähnlich wie bei **Forschungsfrage 3**, sicherzustellen, dass die worteinbettungsbasierte Abkürzungsauflösung unter den bestmöglichen Bedingungen evaluiert wird, dient **Forschungsfrage 4** dazu, das bessere der beiden Worteinbettungsverfahren für die Evaluation des Abkürzungsauflösungsverfahrens auszuwählen.

- **Forschungsfrage 5:** Auf welchen Wissensquellen sollte ein Worteinbettungsmodell trainiert werden, damit ein darauf aufbauendes Abkürzungsauflösungsverfahren eine möglichst hohe Präzision erzielt?

Um die in Abschnitt 5.4.4.3 gestellte Frage zu beantworten, ob ein möglichst großes Worteinbettungsmodell, oder ein möglichst auf die Domäne des Quelltextes spezialisiertes Worteinbettungsmodell zielführender für die worteinbettungsbasierte Abkürzungsauflösung ist, werden im Rahmen von **Forschungsfrage 5** die in Abschnitt 6.2.1 vorgestellten Worteinbettungsmodelle gegeneinander evaluiert, sodass das Beste für die Evaluation des worteinbettungsbasierten Abkürzungsauflösungsverfahrens eingesetzt werden kann.

- **Forschungsfrage 6:** Wie präzise ist die ausschließliche Nutzung der Worteinbettungs-Ähnlichkeitsfunktion für die Auswahl des besten Auflösungskandidaten im Vergleich zu den Verfahren von Alatawi et al.?

In Abschnitt 6.2.2 wurden zwei Möglichkeiten vorgestellt, wie Worteinbettungen zur Abkürzungsauflösung eingesetzt werden könnten: entweder als eigenständiges Verfahren, oder ergänzend zu einem existierenden. **Forschungsfrage 6** dient dazu zu beantworten, ob die hier vorgeschlagene Verwendung von Worteinbettungen bei der Abkürzungsauflösung, im Verhältnis zum Stand der Forschung, welcher durch die Verfahren von Alatawi et al. repräsentiert wird, eine brauchbare Methode zur Abkürzungsauflösung darstellt.

- **Forschungsfrage 7:** Wie präzise ist die Mehrdeutigkeitsauflösung nach Auflösung mit den Verfahren von Alatawi et al., mittels worteinbettungsbasierter Entscheidung, im Vergleich zur willkürlichen Auswahl eines der gleichwertig besten Kandidaten?

Forschungsfrage 7 dient dazu zu evaluieren, ob der hier vorgeschlagene, worteinbettungsbasierte Ansatz erfolgreich ergänzend zu den Verfahren von Alatawi et al. eingesetzt werden kann. Selbst ein Verfahren, welches nicht für sich allein genommen zu erfolgreicher Abkürzungsauflösung führt, könnte erfolgreich zum Ausgleich einer bestimmten Schwäche eines anderen Verfahrens, hier der Mehrdeutigkeitsauflösung bei Alatawi et al., eingesetzt werden.

Um die Validität der Beobachtungen zu gewährleisten, welche diese Forschungsfragen beantworten sollen, müssen zunächst folgende Fragen bezüglich der als Referenzwert und Grundlage der Innovationen genutzten Verfahren von Alatawi et al. beantwortet werden, um die Ausgangssituation zu klären:

- **Ausgangsfrage 1:** Wie präzise ist die Imitation der Verfahren von Alatawi et al. im Vergleich zur originalen Implementierung?

Um einschätzen zu können, inwieweit Ergebnisse, welche mittels der imitierten Implementierung der Verfahren von Alatawi et al., und der darauf aufbauenden Innovationen, erzielt wurden, auf die originale Implementierung der Verfahren von Alatawi et al., und somit auf den aktuellen Stand der Forschung, übertragbar sind, muss, durch die Beantwortung von **Ausgangsfrage 1**, das Verhältnis zwischen der Imitation und dem Original geklärt werden.

- **Ausgangsfrage 2:** Wie groß ist der Einfluss des Fehlers beim Berechnen von Quelltext-Bigramm-Wahrscheinlichkeiten, welchen Alatawi et al. bei ihrer Implementierung gemacht haben?

In Abschnitt 7.2 wird von einem Fehler in der Implementierung von Alatawi et al. berichtet. Es ist einerseits nicht wünschenswert, diesen fortzuführen, andererseits muss geklärt werden, ob etwaige Verbesserung, in einem auf die Ansätze von Alatawi et al. aufbauenden Verfahren, tatsächlich von der Innovation herrühren, oder nur von der Korrektur eines Fehlers im ursprünglichen Verfahren. Um dies einschätzen zu können wird **Ausgangsfrage 2** beantwortet.

- **Ausgangsfrage 3:** Wie präzise sind die Verfahren von Alatawi et al. auf anderen Beispielquelltexten?

Da die hier vorgestellten Verfahren auf verschiedenen Beispielquelltexten evaluiert werden, und die Ergebnisse dieser Evaluationen ins Verhältnis zum aktuellen Stand der Forschung und zu den Verfahren, auf welchen die Innovationen aufbauen, gestellt werden sollen, ist es notwendig **Ausgangsfrage 3** zu beantworten, damit entsprechende Vergleichswerte vorliegen.

9.1 Präzisionsbegriff in dieser Arbeit

Bei binären Klassifizierungsproblemen ist folgender Begriff von Genauigkeit und Präzision üblich: Wird zwischen zwei Klassen A und B unterschieden, beschreibt Genauigkeit die Zahl der Elemente einer Menge, welche richtigerweise A oder B zugeordnet wurden, geteilt durch die Gesamtzahl der Elemente in der Menge. Die Präzision beschreibt, wenn A die beobachtete Klasse ist: Von allen Elementen, welche als A klassifiziert wurden, wie viele sind A zugehörig? Der Begriff Ausbeute hingegen beschreibt, wie viele der Elemente, welche A zugehörig sind, auch tatsächlich als A klassifiziert wurden. Das F_1-Maß zuletzt beschreibt das harmonische Mittel von Präzision und Ausbeute. Die Interpretation dieses Maßes hängt stark vom Kontext ab, allgemein dient es aber dazu eine ausgeglichene Bewertung beider Gütemaße, Präzision und Ausbeute, in einer einzelnen Zahl auszudrücken.

Für die Evaluation eines Verfahrens zur Auflösung von abgekürzten Quelltextbezeichnern, gegeben eine ideale Musterlösung, welche alle Vorkommnisse jedes Quelltextbezeichners inklusive Quelltextdatei- und Zeile enthält, und eine zweite Musterlösung gleicher Qualität, welche nur die abgekürzten Quelltextbezeichner enthält, könnten all diese Maße unabhängig voneinander bestimmt werden. Dann würde die Genauigkeit bezüglich aller Bezeichner beschreiben, wie viele Vorkommnisse von Bezeichnern richtig aufgelöst und richtigerweise nicht aufgelöst wurden, geteilt durch die Zahl aller Vorkommnisse von Bezeichnern. Die Genauigkeit bezüglich abgekürzter Quelltextbezeichner würde analog die Zahl aller richtig aufgelösten Vorkommnisse von abgekürzten Quelltextbezeichnern, geteilt durch die Zahl aller Vorkommnisse von abgekürzten Quelltextbezeichnern beschreiben. Die Präzision könnte als die Zahl richtig aufgelöster Vorkommnisse von Quelltextbezeichnern, geteilt durch die Zahl gefundener und bearbeiteter Vorkommnisse von Quelltextbezeichnern errechnet werden, und erneut als Präzision bezüglich aller Bezeichner und nur der abgekürzten Bezeichner unterteilt werden. Die Ausbeute wäre in diesem Fall identisch mit der Genauigkeit. Werden alle Bezeichner gefunden, wären Genauigkeit, Präzision und Ausbeute jeweils identisch.

Es liegt allerdings, wie in Abschnitt 8.2 beschrieben, leider nur eine solche ideale Musterlösung vor, welche nur vier Quelltextdateien des *Dronology*-Projektes abdeckt. Eine weitere Musterlösung zu *Dronology* enthält zwar alle abgekürzten Bezeichner aus *Dronology*, allerdings ohne Verweis auf Quelltextdatei oder -Zeile. Da das Auflösungswerkzeug grundsätzlich jedes Vorkommnis jedes Quelltextbezeichners findet, ist der fehlende Verweis auf einzelne Vorkommnisse in der Musterlösung aber nur problematisch, wenn zwei Vorkommnisse des gleichen Bezeichners an unterschiedlichen Orten unterschiedlicher Auflösung bedürfen. Da, davon ausgehend, dass alle Vorkommnisse von Quelltextbezeichnern

gefunden werden, die Begriffe der Genauigkeit, Präzision und Ausbeute für diese beiden Musterlösungen identisch sind, sind sie in diesem Kontext austauschbar. Da aber für die weiteren Musterlösungen keine Vollständigkeit garantiert werden kann, ist dort das einzige Maß, über das mit Sicherheit eine Aussage getroffen werden kann, die Präzision. Die Begriffe Genauigkeit und Ausbeute könnten allenfalls mit der Musterlösung als Referenzwert bewertet werden, was der Intention dieser Begriffe nicht gerecht würde. Es ist also sinnvoll, hier grundsätzlich von Präzision zu sprechen, da diese die beobachtete Güte am besten widerspiegelt, und nur für bestimmte Musterlösungen willkürlich einen anderen der dort gleichbedeutenden Begriffe zu verwenden, nur vom Wesentlichen ablenken würde. Insbesondere, wenn eine Auflösung eines Quelltextbezeichners, welche diesen unverändert lässt, als eine Entscheidung des Auflösungsverfahrens gewertet wird und nicht als ein Nichtbehandeln dieses Bezeichners, was der technischen Umsetzung aller hier vorgestellten Verfahren entspricht, ist es zutreffender, auch im Bezug auf diese Bezeichner, von Präzision zu sprechen. Auch enthalten alle Musterlösungen, bis auf eine, nur abgekürzte Quelltextbezeichner. Es wird also in der gesamten Evaluation, außer Abschnitt 9.3, die Präzision bezüglich Abkürzungen behandelt.

Auch wird die Präzision je abgekürztem Quelltextbezeichner in dieser Arbeit grundsätzlich als das relevantere Qualitätsmaß angesehen. Zum einen ist in vielen Software-Projekten ein Großteil der Quelltext-Bezeichner nicht abgekürzt, die Zahl *echt negativer* Ergebnisse, also richtigerweise nicht aufgelöster Bezeichner, würde also die Aussagekraft der Genauigkeit schmälern. Selbst ein Verfahren, welches keinen Bezeichner verändert, könnte eine scheinbar hohe Genauigkeit aufweisen. Zum anderen hängt der Nutzen eines bezüglich allen oder nur bezüglich abgekürzten Bezeichnern präzisen Verfahrens vom Anwendungsfall ab. Wenn das Abkürzungsauflösungsverfahren nur Anwendung findet, wenn bestimmten Bezeichnern andernfalls kein Sinn entnommen werden kann, ist die Präzision bezüglich abgekürzten Bezeichnern das angebrachtere Maß. Im Rahmen des *INDIRECT*-Projekts könnte die Abbildung des Quelltextes auf die Anforderungen beispielsweise genau in den Quelltextbezeichnern automatisch Abkürzungen auflösen, die zuvor nicht ohne diesen Schritt auf eine Anforderung abgebildet werden konnten. Auch andere Informationsrückgewinnungs-Verfahren könnten den Einsatz von automatischer Abkürzungsauflösung auf die Fälle beschränken, in denen sonst keine Information gewonnen wird.

Zudem wird die Präzision der Verfahren nicht nur anhand der Quelltextbezeichner, sondern auch anhand der Token berechnet. Hierbei werden einzelne Abkürzungen oder Token, welche entweder als eigenständiger Bezeichner oder als Segment eines zusammengesetzten Quelltextbezeichners auftreten, betrachtet. Auch hier werden nur solche Segmente betrachtet, welche in Bezeichnern vorkommen, die wiederum in der Musterlösung Erwähnung finden. Ein solches, gegebenenfalls abgekürztes Segment gilt als richtig aufgelöst, wenn die im Auflösungsverfahren ausgewählte, gegebenenfalls zusammengesetzte Auflösung am Index dieses Segmentes identisch ist mit dem Segment am gleichen Index in der Musterlösung. In Beispiel 9.1 wurde eines von zwei Segmenten richtig aufgelöst, bezüglich dieses Bezeichners beträgt die Präzision je Token also 0,5.

Beispiel 9.1: Evaluation des Auflösungsverfahrens je Token

- Bezeichner: *dm*
- Auflösung laut Auflösungsverfahren: *display, may*
- Auflösung laut Musterlösung: *display, metrics*
- Es wurde also eines von zwei Token richtig aufgelöst.

Tabelle 9.1: Präzision der Verfahren von Alatawi et al., in Original und Imitation, bezogen auf richtig aufgelöste abgekürzte Bezeichner aus der Musterlösung, erhoben auf dem Beispielquelltext von Alatawi et al. „NLR“ meint hier die N-Gramm-Grammatiken aus natürlichsprachlichen Quellen, „SCR“ die aus Software-basierten Quellen. Hier wird der Ort des abgekürzten Bezeichners im Quelltext berücksichtigt.

			Präzision
Unigramm	NLR	Original	0,7083
		Imitation	0,5714
	SCR	Original	0,7083
		Imitation	0,5934
Bigramm	NLR	Original	0,7083
		Imitation	0,5384
	SCR	Original	0,7917
		Imitation	0,6703

9.2 Vergleich der originalen und der imitierten Implementierung der Verfahren von Alatawi et al.

Wie bereits in Kapitel 6 begründet, werden die Uni- und Bigramm-basierten Verfahren von Alatawi et al. [AXX17] [AXY18] sowohl als Referenz, als auch als Grundlage für die hier vorgestellten Innovationen verwendet, und dazu in Java im Rahmen des *INDIRECT*-Projekts [Hey19] nach implementiert. Dabei sind die Ergebnisse des Originals und der Imitation nicht ganz identisch. Um das imitierte Verfahren dennoch als Grundlage und Referenzwert für die neuen in dieser Arbeit vorgeschlagenen Ansätze verwenden zu können, dient dieser Abschnitt der Beantwortung der **Ausgangsfrage 1:** Wie präzise ist die Imitation der Verfahren von Alatawi et al. im Vergleich zur originalen Implementierung?

Wie in Abschnitt 7.2 erklärt, wurden im Original die Abkürzungen direkt aus der Musterlösung gewonnen, während sie in der Imitation aus dem Quelltext ausgelesen werden. So löst das Original alle 96 Abkürzungen auf, während die Imitation nur 91 abgekürzte Bezeichner findet, was konkreter in Abschnitt 8.2 erklärt wird. Wie in Tabelle 9.1 zu sehen ist, löst die originale Implementierung des Unigramm-basierten Verfahrens von Alatawi et al. [AXX17], bei Verwendung der auf natürlicher Sprache basierenden Unigramm-Grammatik für die Bestimmung der Auftretensahrscheinlichkeit der Auflösungskandidaten, 68 Abkürzungen von den 96 aufgelösten Abkürzungen aus der Musterlösung richtig auf, und erreicht damit eine Präzision von 0,7083. Die imitierte Implementierung löst, von den 91 als Quelltextbezeichner gefundenen Abkürzungen aus der Musterlösung, bei gleicher Konfiguration des Verfahrens, 52 richtig auf, und erreicht damit eine Präzision von 0,5744. Warum die beiden Verfahren unterschiedlich viele der abgekürzten Bezeichner aus der Musterlösung finden, wird in Abschnitt 8.2 erklärt.

Dass die Imitation, je nach Verfahren, um zwischen 11,49 und 16,99 Prozentpunkte weniger präzise ist, kann zum einen daran liegen, dass, in beiden Implementierungen, im Fall von mehreren gleichwertig besten Auflösungen, eine willkürlich ausgewählt wird. Zum anderen lässt sich nicht ausschließen, dass bei der imitierten Implementierung Fehler passiert sind, welche der Präzision der Verfahren schaden.

Bei Verwendung der aus Software-basierten Quellen gewonnenen Unigramm-Grammatik, für das gleiche Verfahren, löst das Original von Alatawi et al. erneut 68 Abkürzungen

richtig auf, und erreicht damit erneut eine Präzision von 0,7083. Gleiches gilt für die Verwendung des natürlichsprachlichen Datensatzes beim Einsatz des Bigramm-basierten Verfahrens. Die Imitation hingegen erreicht mit 54 richtigen von 91 aufgelösten Bezeichnern, bei Verwendung der Software-basierten Grammatik im Unigramm-basierten Verfahren, eine Präzision von 0,5934. Bei Verwendung des Bigramm-basierten Verfahrens und der natürlichsprachlichen Bigramm-Grammatik erreicht die Imitation 49 richtige von 91 Auflösungen, und damit eine Präzision von 0,5384.

Dass die Imitation bei diesen Konfigurationen jeweils unterschiedlich abschneidet, während das Original bei diesen drei Konfigurationen gleich gut abschneidet, kann zum einen, ebenso wie beim Präzisionsunterschied zwischen Original und Imitation, an der willkürlichen Auswahl unter den gleichwertig besten Auflösungskandidaten liegen. Zum anderen ist es auch beim Original so, dass diese drei Konfigurationen zwar gleich viele Abkürzungen richtig auflösen, allerdings unterschiedliche. Davon sind auch Abkürzungen betroffen, welche von der Imitation nicht als Bezeichner gefunden wurden. Zum anderen können etwaige Fehler oder Unterschiede bei der Imitation der Implementierung dazu führen, dass der im Original nur qualitative Unterschied zwischen den Ergebnissen je Konfiguration, in der Imitation auch quantitative Auswirkungen hat.

Der deutliche Sprung, welcher im Original zwischen den schon genannten Verfahren und dem Bigramm-basierten Verfahren auf dem Software-basierten Datensatz besteht, tritt auch in der Imitation auf, wie in Tabelle 9.1 zu sehen ist. Konkret erreicht das Original mit 76 richtigen Auflösungen von 96 Beispielen aus der Musterlösung eine Präzision von 0,7917, während die Imitation mit 61 richtigen von 91 gefundenen Auflösungen der Abkürzungen aus der Musterlösung eine Präzision von 0,6703 erreicht. Somit ist das Bigramm-basierte Verfahren, unter Nutzung des Software-basierten Datensatzes in der originalen Implementierung, um 8,34 Prozentpunkte, beziehungsweise 11,77% präziser als alle anderen. Auf dem imitierten Quelltext ist es zwischen 13,19 Prozentpunkte, beziehungsweise 24,50%, und 7,69 Prozentpunkte, beziehungsweise 12,96%, präziser als die anderen Verfahren. Gerade der Sprung um 12,96% betrifft den Unterschied des imitierten Quelltextes, auf Basis des Software-basierten Datensatzes, zwischen der Verwendung des Unigramm- und des Bigramm-basierten Verfahrens, und diese Differenz kommt dem Original recht nahe. Es scheint also, als sei der Unterschied zwischen den Unigramm- und Bigramm-basierten Verfahren treffend imitiert worden.

Diese Unterschiede stellen die Antwort auf die **Ausgangsfrage 1** dar, und sollten bei der Evaluation der hier vorgestellten Innovationen berücksichtigt werden, um die Aussagekraft der Ergebnisse gegenüber dem Verfahren von Alatawi et al. nicht zu überschätzen. Allerdings bietet die Imitation dennoch sowohl ein Grundlage, um weitere Verfahren darauf aufzubauen, als auch einen Referenzwert, der wenigstens relative Aussagen über die Leistung anderer Verfahren ermöglicht.

In der imitierten Implementierung der Verfahren von Alatawi et al. werden auch die richtigen Auflösungen je Token ausgegeben. Die 91 gefundenen abgekürzten Bezeichner enthalten insgesamt 221 Token. In Tabelle 9.2 ist zu sehen, dass die Präzision je Token auf dem Beipsielquelltext von Alatawi et al. allgemein höher ist als die je Bezeichner. Das rührt daher, dass, wie in Beispiel 9.1, einige einzelne Segmente aus zusammengesetzten Bezeichnern zwar zu den richtigen Token aufgelöst werden, der Bezeichner aber letztendlich falsch aufgelöst wird.

9.2.1 Einfluss des Fehlers in der Implementierung von Alatawi et al.

Wie in Abschnitt 7.2 erklärt wird, enthält die originale Implementierung des Verfahrens von Alatawi et al. einen Fehler, aufgrund dessen für die Bigramm-Grammatiken von Quelltextbezeichnern die gleiche Statistik geladen wird, wie für Quelltext-bezogene Unigramme.

Tabelle 9.2: Präzision des imitierten Verfahren von Alatawi et al., bezogen auf richtig aufgelöste abgekürzte Token aus Bezeichnern aus der Musterlösung, erhoben auf dem Beispielquelltext von Alatawi et al. „NLR“ meint hier die N-Gramm-Grammatiken aus natürlichsprachlichen Quellen, „SCR“ die aus Software-basierten Quellen. Hier wird der Ort des abgekürzten Bezeichners im Quelltext berücksichtigt.

		Präzision (Token)
UNI	NLR	0,7330
	SCR	0,7376
BI	NLR	0,7240
	SCR	0,7828

Tabelle 9.3: Präzision des imitierten Verfahrens von Alatawi et al., bezogen auf richtig aufgelöste abgekürzte Bezeichner aus der Musterlösung, erhoben auf dem Beispielquelltext von Alatawi et al. „NLR“ meint hier die N-Gramm-Grammatiken aus natürlichsprachlichen Quellen, „SCR“ die aus Software-basieren Quellen. Hier wird der Ort des abgekürzten Bezeichners im Quelltext nicht berücksichtigt.

		Präzision
UNI	NLR	0,2439
	SCR	0,2773
BI	NLR	0,2516
	SCR	0,3427

Dieser Fehler ist der Grund für **Ausgangsfrage 2:** Wie groß ist der Einfluss des Fehlers beim Berechnen von Quelltext-Bigramm-Wahrscheinlichkeiten, welchen Alatawi et al. bei ihrer Implementierung gemacht haben? Tatsächlich macht das Berechnen richtiger Bigramme aus dem Quelltext allerdings keinen Unterschied bei der Zahl richtig aufgelöster abgekürzter Quelltextbezeichner im Beispielquelltext von Alatawi et al., weder bei Einsatz des natürlichsprachlichen, noch der Software-basierten Bigramm-Grammatik für die Auflösungskandidaten. Ob beim imitierten Verfahren der Fehler wiederholt wurde oder nicht, die Ergebnisse sind die Selben und somit die in Tabelle 9.1 und Tabelle 9.2 aufgeführten. Bei der Evaluation auf allen weiteren Beispielquelltexten werden die Quelltextbezogenen Uni-, Bi- und Trigramm-Grammatiken, welche als Bedingung für die bedingte Wahrscheinlichkeit der Uni-, Bi- und Trigramme in den Auflösungskandidaten dienen, während der Ausführung des Auflösungswerkzeuges aus dem Quelltext berechnet. Der Fehler von Alatawi et al. wird für diese also nicht wiederholt.

9.2.2 Auflösung aller Vorkommnisse abgekürzter Bezeichner auf dem Beispielquelltext von Alatawi et al.

In Abschnitt 8.2 wird erläutert, dass die Musterlösung von Alatawi et al. auf bestimmte Vorkommnisse von bestimmten abgekürzten Bezeichnern verweist, indem sie die Quelltextdatei und -Zeile, in welcher diese auftreten, berücksichtigt. In der imitierten Implementierung der Verfahren von Alatawi et al. kann dies allerdings deaktiviert werden. Ist diese Berücksichtigung des Ortes eines bestimmten Vorkommnisses jedes abgekürzten Bezeichners in der Musterlösung deaktiviert, wird stattdessen jedes Vorkommnis jedes dieser Bezeichner berücksichtigt. Somit werden die abgekürzten Bezeichner, die gegebenenfalls mehrfach auftreten und mehrfach aufgelöst werden, auch mehrfach in der Evaluation ge-

wertet. Diese Evaluation ist also vergleichbarer mit der, die anhand der Musterlösungen für *a2ps*, *which* und der für den gesamten Quelltext von *Dronology* vorgenommen wird.

Insgesamt werden 779 Vorkommnisse von 80 verschiedenen abgekürzten Quelltextbezeichnern gefunden. Die 80 anstatt 91 verschiedenen Bezeichner rühren von den in Abschnitt 8.2 erwähnten elf Einträgen der Musterlösung her, in welchen für einen Bezeichner zwei Vorkommnisse eingetragen wurden. Wie in Tabelle 9.3 zu sehen ist, ist die Präzision des Verfahrens, gemessen an allen Vorkommnissen der abgekürzten Bezeichner aus der Musterlösung, weit schlechter, als wie in Tabelle 9.1 beschrieben, an je nur einem Vorkommnis jedes Bezeichners.

9.2.3 Imitation der Verfahren von Alatawi et al. auf anderen Beispielquelltexten

Die Musterlösungen, welche für *a2ps* und *which* vorliegen, und die, welche über alle Abkürzungen im Quelltext von *Dronology* produziert wurde, bilden jeden abgekürzten Bezeichner allgemein auf eine Auflösung ab, und enthalten keine Information über den Ort des Auftretens. Mehrfach auftretende Bezeichner werden somit mehrfach aufgelöst, wobei sie gegebenenfalls im jeweiligen Kontext unterschiedlich aufgelöst werden können, und werden in der Evaluation mehrfach gewertet. Somit wird eine Aussage über die Auflösung aller Vorkommnisse abgekürzter Bezeichner getroffen, und nicht nur jeweils über die Auflösung eines Vertreters von mehreren Vorkommnissen eines Bezeichners. Genauer wird darauf in Abschnitt 8.2 und Abschnitt 9.1 eingegangen.

Die Evaluation der Verfahren von Alatawi et al. auf diesen Beispielquelltexten dient der Beantwortung von **Ausgangsfrage 3:** Wie präzise sind die Verfahren von Alatawi et al. auf anderen Beispielquelltexten? Bei der Evaluation der Verfahren von Alatawi et al., unter Nutzung der imitierten Implementierung, auf den anderen Beispielprojekten, konkreter *Dronology* [ND21], *a2ps* [a2p21] und *which* [Woo21] in Tabelle 9.4, fallen zwei Dinge besonders auf: Erstens liefert der Versuch, die Abkürzungen in den Quelltextbezeichnern von *Dronology* aufzulösen, allgemein schlechte Ergebnisse, und zweitens sind die Ergebnisse je Beispielprojekt nahezu unabhängig von der Konfiguration des Auflösungswerkzeuges. Bei Betrachtung des Quelltextes von *Dronology* und Tabelle 9.5 fällt auf, dass *Dronology* allgemein recht wenige Abkürzungen enthält, und dass die Präzision bei der Auflösung aller Bezeichner in etwa der Präzision der Auflösung der Abkürzungen in *a2ps* oder *which* entspricht. Auflösungen aus der *a2ps*-Musterlösung enthalten im Schnitt 1,33 Token. Die Auflösungen in der Musterlösung für *Dronology* enthalten im Schnitt 2,52 Token. Es kommen also im Schnitt in einer Auflösung mehr Token vor, von denen nur eines falsch aufgelöst zu werden braucht, damit der gesamte Bezeichner falsch aufgelöst wird. Darüber hinaus werden aber auch die einzelnen Token seltener richtig aufgelöst. Die Unabhängigkeit oder annähernde Unabhängigkeit der Ergebnisse von der Konfiguration ist ein wenig verwunderlich, da es auf dem von Alatawi et al. mitgelieferten Beispiel doch deutlichere Unterschiede gab, allerdings zeigt auch der Vergleich des Originals mit der Imitation, dass diese Unterschiede gewissen Schwankungen unterliegen, in welchen teils ein Bezeichner, welcher in einer Konfiguration falsch aufgelöst wurde, in einer anderen richtig aufgelöst wird, aber dies eben für einen anderen Bezeichner umgekehrt ist. Somit ist es denkbar, insbesondere angesichts der größeren Zahl betrachteter Bezeichner, dass diese Schwankungen sich amortisieren. Bei der Auflösung der Abkürzungen in *a2ps* und *which* schwankt zumindest die Präzision bezüglich richtig aufgelöster Token, nicht aber die Zahl richtig aufgelöster Bezeichner. Bei *Dronology* sind beide konstant. In Abschnitt C.2 und Beispiel C.2 sind einige der richtigen und falschen Auflösungen aus dem *Dronology*-Quelltext zu sehen. In diesen beiden Listen fällt auf, dass einige der abgekürzten Bezeichner, welche mindestens einmal richtig aufgelöst wurden, andernorts falsch aufgelöst wurden. Dies zeigt die Schwäche von Musterlösungen wie der von Alatawi et al., welche jeden abgekürzten Bezeichner nur einmal in einem vorausgewählten Kontext enthalten, wie auch in Abschnitt 9.2.2 gezeigt

Tabelle 9.4: Präzision der Verfahren von Alatawi et al. auf diversen Beispielquelltexten. „NLR“ meint hier die N-Gramm-Grammatiken aus natürlichsprachlichen Quellen, „SCR“ die aus Software-basierten Quellen. Hier wird der Ort des abgekürzten Bezeichners im Quelltext nicht berücksichtigt.

			Kontext	Methodenaufrufe (bidirektional)	Präzision (Bezeichner)	Präzision (Token)
Dronology	Unigramm	NLR	Klasse	nein	0,0511	0,0671
	Unigramm	SCR	Klasse	nein	0,0511	0,0671
	Unigramm	NLR	Methode	nein	0,0511	0,0671
	Unigramm	NLR	Methode	ja	0,0511	0,0671
	Bigramm	NLR	Klasse	nein	0,0511	0,0671
	Bigramm	SCR	Klasse	nein	0,0511	0,0671
a2ps	Unigramm	NLR	Klasse	nein	0,3667	0,2551
	Unigramm	SCR	Klasse	nein	0,3678	0,2556
	Unigramm	SCR	Methode	nein	0,3678	0,2556
	Bigramm	NLR	Klasse	nein	0,3678	0,2544
	Bigramm	SCR	Klasse	nein	0,3678	0,2544
which	Unigramm	NLR	Klasse	nein	0,3938	0,2678
	Unigramm	SCR	Klasse	nein	0,3938	0,2684
	Unigramm	SCR	Methode	nein	0,3938	0,2684
	Bigramm	NLR	Klasse	nein	0,3938	0,2684
	Bigramm	SCR	Klasse	nein	0,3938	0,2657

Die Musterlösung für Abkürzungen in *Dronology* enthält 120 Einträge, davon wurden 113 insgesamt 998 mal gefunden.
Die Musterlösung für Abkürzungen in *a2ps* enthält 205 Einträge, davon wurden 177 insgesamt 1.846 mal gefunden.
Die Musterlösung für Abkürzungen in *which* enthält 474 Einträge, davon wurden 299 insgesamt 2.242 mal gefunden.

wird. Diese Fälle sind offenbar ein Grund dafür, dass die Evaluationen anhand aller anderen Musterlösungen niedrigere Präzisionen aufweisen, als die Evaluationen anhand der Musterlösung von Alatawi et al. Das unterschiedliche Auflösen von zwei Vorkommnissen des gleichen Bezeichners kann nur am Kontext liegen, da dieser die einzige Variable ist. Ist dieser Kontext für einige bestimmte Vorkommnisse bestimmter abgekürzter Bezeichner besonders günstig, während er für die anderen Vorkommnisse abgekürzter Bezeichner nicht auf die richtige Auflösung schließen lässt, kann dies unter Umständen einen größeren Einfluss haben, als die verschiedenen Konfigurationen der Verfahren, was die geringen, beziehungsweise für *Dronology* nicht vorhandenen, Schwankungen in der Präzision erklären würde.

Auch wurde evaluiert, ob es einen Unterschied macht, die Methode, anstelle der Klasse, als Kontext zur Kandidatensuche heranzuziehen. Dass dies nicht der Fall ist, ist allein dadurch naheliegend, dass die Verfahren von Alatawi et al. Auflösungskandidaten bevorzugen, welche möglichst in der Nähe des Bezeichners auftreten. Ebenso verhält es sich mit der Berücksichtigung von Methodenaufrufen bei der Suche nach Auflösungskandidaten, welche ebenfalls keinen Unterschied macht. Eine Evaluation unter Verwendung von noch kleineren Kontexten als der Methode wird hier nicht aufgelistet, da noch kleinere Kontexte bezüglich aller Verfahren, wie in Abschnitt 6.3 erwartet, nur schlechtere Ergebnisse hervorgebracht haben, und ihre ausführliche Behandlung deshalb für überflüssig erachtet wird.

9.3 Nichtauflösen von Bezeichnern, welche in einem Wörterbuch auftreten

Eine in Abschnitt 6.4 vorgestellte Option, welche für alle hier vorgestellten Verfahren konfigurierbar ist, ist das Überspringen von Quelltextbezeichnern, welche in einem Wörterbuch [Web21] auftreten. Um zu evaluieren, ob diese Option im allgemeinen eingesetzt werden sollte, wird **Forschungsfrage 1** gestellt: Wie groß ist der Einfluss des Auflösens, beziehungsweise des Nichtauflösens, von Wörterbuch-Wörtern auf die Präzision der Verfahren von Alatwi et al.? Die imitierte Implementierung des Unigramm-Verfahrens von Alatawi et al. löst, bei Verwendung des natürlichsprachlichen Datensatzes, 6.216 Bezeichner auf, obwohl diese im Wörterbuch auftreten. Diese könnten unter der Annahme, dass ein Wort, welches im Wörterbuch vorkommt, keine Abkürzung ist und keiner Auflösung bedarf, übersprungen werden. Allerdings treten 15 Abkürzungen aus der Musterlösung im Wörterbuch auf, so zum Beispiel „at“, „as“ oder „it“. Bei der Unigramm-basierten Auflösung die Bezeichner, welche im Wörterbuch stehen, zu überspringen, reduziert, bei Einsatz der auf natürlicher Sprache basierenden Unigramm-Grammatik, die Zahl richtiger Auflösungen der abgekürzten Bezeichner von 52 auf 43, und reduziert damit die Präzision von 0,5714 auf rund 0,4725.

Um die Zahl *falsch positiver* Auflösungen, also von Bezeichnern, welche aufgelöst wurden, obwohl sie keine Abkürzung darstellen, und *echt negativer*, also solcher, welche richtigerweise nicht aufgelöst wurden, abschätzen zu können, wurde eine weitere Musterlösung für das *Dronology*-Projekt angelegt. Diese beinhaltet nicht nur abgekürzte Bezeichner, sondern alle Quelltextbezeichner. Eine solche Musterlösung ist extrem aufwändig zu produzieren, weswegen nur eine angelegt wurde, welche nur vier Quelltextdateien aus *Dronology* abdeckt. Zudem verweist diese Musterlösung, im Gegensatz zu allen anderen, bis auf die von Alatawi et al. bereitgestellte, auch auf die Quelltextdatei und -Zeile eines Bezeichners, sodass einzelne Vorkommnisse der Bezeichner gewertet werden können. Um die Präzision auf allen Bezeichnern direkt mit der Präzision auf abgekürzten Bezeichnern vergleichen zu können, wurde eine Kopie angelegt, welche ebenfalls nur die gleichen vier Quelltextdateien abdeckt und ebenso die Quelltextdatei und Quelltextzeile eines Bezeichners berücksichtigt,

Tabelle 9.5: Unigramm-basiertes Verfahren von Alatawi et al., unter Verwendung der natürlichsprachlichen Unigramm-Grammatik, evaluiert auf vier Quelltextdateien des *Dronology* Beispielprojektes, gegen die Musterlösung, welche den Quelltextort eines Auftretens eines Bezeichners berücksichtigt.

Musterlösung (alle)	Musterlösung (abgekürzt)	richtig (alle)	richtig (abgekürzt)	Präzision (alle)	Präzision (abgekürzt)
498	77	182	10	0,3655	0,1169

Tabelle 9.6: Unigramm-basiertes Verfahren von Alatawi et al., unter Verwendung der natürlichsprachlichen Unigramm-Grammatik, evaluiert auf vier Quelltextdateien des *Dronology* Beispielprojektes, gegen die Musterlösung, welche den Quelltextort eines Auftretens eines Bezeichners berücksichtigt, unter Ausschluss von Wörterbuch Wörtern

Musterlösung (alle)	Musterlösung (abgekürzt)	richtig (alle)	richtig (abgekürzt)	Präzision (alle)	Präzision (abgekürzt)
498	77	359	4	0,7209	0,0519

aber nur abgekürzte Bezeichner behandelt. Dennoch wurde diese Musterlösung nicht für die Evaluation aller Verfahren eingesetzt, da die Musterlösung, welche die Abkürzungen aus dem gesamten Quelltext von *Dronology* enthält, für aussagekräftiger befunden wurde. Diese Musterlösungen und die diese betreffenden Entscheidungen werden in Abschnitt 8.2 diskutiert.

Wie Tabelle 9.5 zeigt, werden von 498 gefundenen Bezeichnern 182 richtig aufgelöst und gespalten, beziehungsweise richtigerweise nicht aufgelöst oder gespalten, und davon waren 10 abgekürzt. 65 abgekürzte Bezeichner wurden nicht oder falsch aufgelöst, und 249 nicht abgekürzte Bezeichner wurden fälschlicherweise aufgelöst oder falsch gespalten. Es wurde kein einziger der insgesamt 172 gefundenen, zusammengesetzten Bezeichner, davon 37 abgekürzt und 139 nicht abgekürzt, richtig aufgelöst beziehungsweise gespalten. Beispiele hierzu sind in Beispiel C.1 zu finden.

In Tabelle 9.6 wurde das gleiche Verfahren auf dem gleichen Quelltext gegen die gleiche Musterlösung evaluiert, allerdings wurden hier Wörter aus dem Wörterbuch bei der Auflösung ausgeschlossen. Die Präzision bezüglich allen Bezeichnern hat sich dabei nahezu verdoppelt, während die Präzision bezüglich abgekürzter Bezeichner deutlich zurückgegangen ist. Was hier passiert ist, ist, dass weniger natürlichsprachliche Wörter fälschlicherweise aufgelöst wurden, während einige Abkürzungen, welche homonym mit natürlichsprachlichen Wörtern sind, wie „param“, „lat“ oder „alt“, fälschlicherweise unverändert blieben. Dieses Beispiel zeigt deutlich, warum die Präzision bezüglich allen Bezeichnern hier als Qualitätsmaß ungeeignet ist: Die meisten Bezeichner sind nicht abgekürzt, das Verfahren wird also präziser, je weniger Bezeichner, und damit auch je weniger Abkürzungen, aufgelöst werden. Abkürzungen nicht aufzulösen kann für die Bewertung eines Abkürzungsauflösungsverfahrens aber nicht sinnvollerweise als positiv erachtet werden.

9.4 Trigramm-basierte Auflösung

Eine erste der hier vorgestellten Innovationen, gegenüber den von Alatawi et al. vorgeschlagenen Verfahren, ist die Nutzung von Trigrammen für die Bestimmung der Wahrscheinlichkeit von Auflösungskandidaten. Dieser Abschnitt befasst sich mit **Forschungsfrage 2:** Wie hoch ist die Präzision des Trigramm-basierten Verfahrens im Vergleich zu den Verfahren von Alatawi et al.? In Tabelle 9.7 ist zu sehen, dass diese Methode, bei Verwendung

Tabelle 9.7: Trigramm-basiertes Verfahren im Direktvergleich mit Bigramm-basiertem Verfahren. Es wurde jeweils die Software-basierte Uni- und Bigramm-Grammatik verwendet. Die Trigramm-Grammatik stammt aus dem Google N-Gramm-Datensatz.

Quelltext	Verfahren	Präzision (Bezeichner)	Präzision (Token)
Alatawi et al.	Trigramm	0,7033	0,7873
Alatawi et al.	Bigramm	0,6703	0,7828
Dronology	Trigramm	0,0511	0,0671
Dronology	Bigramm	0,0511	0,0671
a2ps	Trigramm	0,3673	0,2537
a2ps	Bigramm	0,3678	0,2544
which	Trigramm	0,3916	0,2660
which	Bigramm	0,3938	0,2657

Tabelle 9.8: Trigramm-, Bigramm- und Unigramm-basiertes Verfahren auf dem Beispielquelltext von Alatawi et al. Die verwendete Uni-, Bi- und Trigramm-Grammatik stammt jeweils aus dem Google N-Gramm-Datensatz.

Verfahren	Präzision (Bezeichner)	Präzision (Token)
Trigramm	0,5055	0,7104
Bigramm	0,5055	0,6833
Unigramm	0,5385	0,7195

der Software-basierten Uni- und Bigramm-Grammatiken [XXA+18], auf allen Beispielprojekten vergleichbare, auf dem Beispielquelltext von Alatawi et al. aber tatsächlich leicht bessere Ergebnisse liefert.

Dass der gewünschte Vorteil nicht immer auftritt, und nicht allzu groß ist, ist schon dadurch naheliegend, dass das Bigramm-basierte Verfahren von Alatawi et al. [AXY18] bereits paarweise alle Bigramme in einem zusammengesetzten Auflösungskandidaten in die Wahrscheinlichkeit des Auflösungskandidaten einbezieht, und somit über die paarweise Bigramm-Wahrscheinlichkeit eine N-Gramm-Wahrscheinlichkeit abgeschätzt wird. Diese abgeschätzte N-Gramm-Wahrscheinlichkeit stattdessen an den auftretenden Trigrammen festzumachen, macht also nur insofern einen Unterschied, wie die Wahrscheinlichkeit der Trigramme von der Wahrscheinlichkeit zweier, in Form von drei Token, nacheinander auftretenden Bigramme abweicht.

Dennoch ist zumindest auf dem von Alatawi et al. bereitgestellten Beispielquelltext ein Vorteil erkennbar. Da die Trigramme nicht aus der gleichen Wissenquelle stammen wie die Uni- und Bigramm-Grammatiken, könnte dies aber vielleicht nicht der Verwendung von Trigramm-Grammatiken an sich zuzuschreiben sein, sondern der Wissensquelle [LML+20], welche für die Erzeugung der Trigramm-Grammatik genutzt wurde. Um diese Hypothese zu prüfen, wurden auch Uni- und Bigramm-Grammatiken aus dem Google Datensatz [LML+20] genutzt, um die Uni-, Bi- und Trigramm-basierten Verfahren unter Nutzung von N-Gramm-Grammatiken zu evaluieren, die alle aus diesem Datensatz stammen. Die Ergebnisse in Tabelle 9.8 lassen allerdings darauf schließen, dass der Google N-Gramm-Datensatz an sich nicht der Grund für die Überlegenheit des Trigramm-basierten Verfah-

Tabelle 9.9: *word2vec*-basiertes Verfahren, evaluiert auf dem Beispielquelltext von Alatawi et al., unter Verwendung des *SO_200*-Worteinbettungsmodells und verschiedenen Ähnlichkeitsfunktionen.

Ähnlichkeitsfunktion	Präzision (Bezeichner)	Präzision (Token)
Naiv-Brachial	0,0879	0,2353
Kontext-Brachial	0,0879	0,2398
Kontext-Wortarten	0,1099	0,2534
Kontext-Aufgeteilt	0,1429	0,3394
Kombinierte-Vektoren	0,1538	0,3394

rens gegenüber dem Bigramm-basierten in Tabelle 9.7 sein kann, da sie allgemein schlechter sind als die Ergebnisse aller drei Verfahren unter Nutzung anderer Datensätze. Trigramm-Grammatiken bleiben also eine vielversprechende Erweiterung des Ansatzes von Alatawi et al.

9.5 Worteinbettungsbasierte Auflösung

In Abschnitt 6.2 werden verschiedene Parameter vorgestellt, welche im Zusammenhang mit worteinbettungsbasierter Abkürzungsauflösung variiert werden können. Um die bestmögliche Verwendung für Worteinbettungen in der Abkürzungsauflösung zu finden, werden in Abschnitt 9.5.1 die verschiedenen vorgeschlagenen Ähnlichkeitsfunktionen gegeneinander evaluiert. In Abschnitt 9.5.2 werden *fastText* und *word2vec* als Worteinbettungsverfahren gegenübergestellt. Schließlich wird in Abschnitt 9.5.3 geprüft, ob ein im voraus trainiertes, umfassendes Worteinbettungsmodell, oder ein explizit für die Abkürzungsauflösung in einem bestimmten Quelltext trainiertes Modell geeigneter sind.

Mit diesen getroffenen Entscheidungen, kann das resultierende worteinbettungsbasierte Abkürzungsauflösungsverfahren in Abschnitt 9.5.4 und Abschnitt 9.5.5 für die verschiedenen in Abschnitt 6.3 vorgeschlagenen Anwendungsfälle auf die Probe gestellt werden.

9.5.1 Ähnlichkeitsfunktion bei Nutzung der Einbettung

Wie in Abschnitt 6.2.3 diskutiert, wird eine Ähnlichkeitsfunktion benötigt, um die durch das Worteinbettungsmodell gegebene Vektordarstellung von Token zur Auflösung von abgekürzten Bezeichnern einzusetzen. Die vorgeschlagenen Ähnlichkeitsfunktionen werden hier, unter Einsatz von *word2vec* als Worteinbettungsverfahren, auf den verschiedenen Beispielprojekten gegeneinander verglichen, um **Forschungsfrage 3** zu beantworten: Welche der vorgeschlagenen Ähnlichkeitsfunktionen für die Abkürzungsauflösung auf Basis von Worteinbettungen liefert die größte Präzision? Für alle Evaluationen wird hierbei die Klasse als Kontext zur Suche nach Auflösungskandidaten verwendet, ohne Rückverfolgung von Methodenaufrufen.

Für alle Beispiele in diesem Abschnitt wurden, wie in Abschnitt 6.2.2 beschrieben, die Kandidaten aus der Kandidatensuche des imitierten Verfahrens von Alatawi et al. gewonnen, ohne dass diese einer Vorauswahl unterzogen wurden. Als Worteinbettungsmodell wird das „Word Embeddings for the Software Engineering Domain“ Modell [ECS18], kurz *SO_200*-verwendet.

In Tabelle 9.9 ist zu sehen, dass die *Naiv-Brachial*-Ähnlichkeitsfunktion, die in Abschnitt 6.2.3.1 beschrieben wurden, einen nicht nennenswerten Nachteil gegenüber der *Kontext-Brachial*-Ähnlichkeitsfunktion aus Abschnitt 6.2.3.2 hat. Dies bekräftigt die Vermutung aus Abschnitt 5.4.4.3, dass der Vergleich der Segmente des Auflösungskandidaten untereinander

Tabelle 9.10: *word2vec*-basiertes Verfahren, evaluiert auf verschiedenen Beispielquelltexten, unter Verwendung des *SO_200*-Worteinbettungsmodells und verschiedenen Ähnlichkeitsfunktionen.

Quelltext	Ähnlichkeitsfunktion	Präzision (Bezeichner)	Präzision (Token)
Dronology	Kontext-Brachial	0,0431	0,0617
Dronology	Kontext-Aufgeteilt	0,0461	0,0603
Dronology	Kombinierte-Vektoren	0,0511	0,0651
a2ps	Kontext-Brachial	0,3651	0,2509
a2ps	Kontext-Aufgeteilt	0,3646	0,2505
a2ps	Kombinierte-Vektoren	0,3629	0,2470
which	Kontext-Brachial	0,3934	0,2660
which	Kontext-Aufgeteilt	0,3934	0,2681
which	Kombinierte-Vektoren	0,3952	0,2675

nicht zur Wahrscheinlichkeit des Auflösungskandidaten im Kontext beiträgt. Gleichzeitig ist es nicht verwunderlich, dass der Unterschied zwischen diesen beiden Ähnlichkeitsfunktionen nur marginal ist, da der Einfluss der wenigen Segmente eines zusammengesetzten Bezeichners gegenüber dutzenden Wörtern im Kontext untergeht.

Außerdem ist in Tabelle 9.9 zu sehen, dass die Nutzung der *Kontext-Wortarten*-Ähnlichkeitsfunktion aus Abschnitt 6.2.3.3 einen leichten Vorteil bringt. Dieser ist allerdings nicht so hoch wie der, welcher durch die *Kontext-Aufgeteilt*-Ähnlichkeitsfunktion aus Abschnitt 6.2.3.4, oder durch die in Abschnitt 6.2.3.5 beschriebene *Kombinierte-Vektoren*-Ähnlichkeitsfunktion erbracht wird. Für die Zukunft könnte der Einsatz eines Wortartenmarkieres, in anderer Form oder in Ergänzung zu anderen Methoden der Kandidatenauswahl, durchaus noch Potential bieten, in der hier dargestellten Verwendung unterliegt er allerdings den Alternativen. Zudem ist die Nutzung des Wortartenmarkierers sehr Rechenzeit-intensiv. Mit einiger Optimierung ließe sich dieser Nachteil sicherlich verringern, aber die schlechteren Ergebnisse bieten dazu aktuell keinen Anlass.

Auf allen anderen Beispielprojekten sind sowohl die *Kontext-Brachial*-Ähnlichkeitsfunktion, die *Kontext-Aufgeteilt*-Ähnlichkeitsfunktion, als auch die *Kombinierte-Vektoren*-Ähnlichkeitsfunktion nahezu gleich gut, wie Tabelle 9.10 zeigt. Bei *Dronology* und *which* ist *Kombinierte-Vektoren* als Ähnlichkeitsfunktion knapp am besten, während gerade diese Ähnlichkeitsfunktion für *a2ps* knapp am schlechtesten ist. Allerdings sprechen die Ergebnisse in Tabelle 9.9, bezüglich des Beispielquelltextes von Alatawi et al., deutlich für *Kombinierte-Vektoren*. Zudem ist diese Ähnlichkeitsfunktion etwas berechnungseffizienter.

9.5.2 *fastText* oder *word2vec*

Dieser Abschnitt dient dazu, **Forschungsfrage 4** zu beantworten: Welches Worteinbettungsverfahren liefert als Grundlage für die hier vorgestellten worteinbettungsbasierten Abkürzungsauflösungsverfahren die höhere Präzision: *fastText* oder *word2vec*? Hierzu wurde das worteinbettungsbasierte Auflösungsverfahren sowohl auf *word2vec*-, als auch auf *fastText*-Modellen ausgeführt.

Zunächst wurden beide worteinbettungsbasierten Auflösungsverfahren auf dem von Alatawi et al. bereitgestellten Beispielquelltext ausgeführt, wobei für beide jeweils ein anderes, vortrainiertes Modell genutzt wurde. Für das *word2vec*-basierte Verfahren wurde das „Word Embeddings for the Software Engineering Domain“ Modell [ECS18], kurz *SO_200*,

Tabelle 9.11: Vergleich zwischen dem *fastText*-basierten Verfahren, unter Einsatz des [Common Crawl] Modells, und dem *word2vec*-basierten Verfahren, unter Einsatz des *SO_200*-Worteinbettungsmodells, auf dem Beispielquelltext von Alatawi et al.

Ähnlichkeitsfunktion	Verfahren	Präzision (Bezeichner)	Präzision (Token)
Kontext-Brachial	*word2vec*	0,0879	0,2398
Kontext-Brachial	*fastText*	0,1099	0,3122
Kontext-Aufgeteilt	*word2vec*	0,1429	0,3394
Kontext-Aufgeteilt	*fastText*	0,1209	0,3258
Kombinierte-Vektoren	*word2vec*	0,1538	0,3394
Kombinierte-Vektoren	*fastText*	0,1319	0,3575

Tabelle 9.12: Vergleich zwischen dem *fastText*-basierten Verfahren und dem *word2vec*-basierten Verfahren, mit Worteinbettungsmodellen, welche auf dem Beispielquelltext selbst trainiert wurden. Die Ähnlichkeitsfunktion ist jedem Fall *Kombinierte-Vektoren*.

Quelltext	Verfahren	Präzision (Bezeichner)	Präzision (Token)
Alatawi et al.	*word2vec*	0,1099	0,3077
Alatawi et al.	*fastText*	0,1319	0,3575
Dronology	*word2vec*	0,0511	0,0632
Dronology	*fastText*	0,0511	0,0767
a2ps	*word2vec*	0,3619	0,2446
a2ps	*fastText*	0,3608	0,2477
which	*word2vec*	0,3943	0,2663
which	*fastText*	0,3934	0,2669

genutzt, für *fastText* ein auf Common Crawl [Cra21] trainiertes und von Facebook bereitgestelltes Modell [Fac21]. Diese werden in Abschnitt 6.2.1 vorgestellt. Wie in Tabelle 9.11 zu sehen ist, kann bei wechselnden Ähnlichkeitsfunktionen keine klare Aussage darüber getroffen werden, welches der beiden Verfahren in dieser Konfiguration tatsächlich besser ist. Allerdings ist hier das allgemein beste Verfahren *word2vec*, mit der auf a kombinierten Vektoren basierenden Ähnlichkeitsfunktion.

Um einen Vergleich zu erreichen, welcher nicht durch verschiedene Wissensquellen für das Training des Worteinbettungsmodells voreingenommen ist, wurden beide Verfahren auf allen Beispielquelltexten mit Modellen evaluiert, welche auf dem Quelltext selbst trainiert wurden. Wie in Tabelle 9.12 zu sehen ist, kann auch hier kein klar besseres Verfahren ermittelt werden. Im folgenden wird allerdings *word2vec* weiterhin genutzt, da dieses eine bessere Laufzeit aufweist.

9.5.3 Wissensquelle für das Worteinbettungsmodell

Um die Frage zu beantworten, ob ein möglichst breites oder ein möglichst spezialisiertes Worteinbettungsmodell besser für die Abkürzungsauflösung mithilfe von Worteinbettungen geeignet ist, behandelt dieser Abschnitt **Forschungsfrage 5:** Auf welchen Wissensquellen

Tabelle 9.13: *word2vec*-basiertes Verfahren, evaluiert auf verschiedenen Beispielquelltexten, mittels verschiedener Worteinbettungsmodelle, unter Nutzung der *Kombinierte-Vektoren*-Ähnlichkeitsfunktion.

Quelltext	Wissensquelle	Präzision (Bezeichner)	Präzision (Token)
Alatawi et al.	Quelltext	0,1099	0,3077
Alatawi et al.	*SO_200*	0,1538	0,3394
Dronology	Quelltext	0,0511	0,0632
Dronology	*SO_200*	0,0511	0,0651
a2ps	Quelltext	0,3619	0,2446
a2ps	*SO_200*	0,3629	0,2470
which	Quelltext	0,3943	0,2663
which	*SO_200*	0,3952	0,2675

Tabelle 9.14: *word2vec*-basiertes Verfahren, evaluiert auf Quelltext von *Dronology*, mittels verschiedener Worteinbettungsmodelle, unter Nutzung der *Kombinierte-Vektoren*-Ähnlichkeitsfunktion.

Wissensquelle	Präzision (Bezeichner)	Präzision (Token)
SO_200	0,0511	0,0651
Dronology Quelltext	0,0511	0,0632
mehrere Drohnen-Programme	0,0481	0,0656
Wikipedia Artikel	0,0401	0,0468

sollte ein Worteinbettungsmodell trainiert werden, damit ein darauf aufbauendes Abkürzungsauflösungsverfahren eine möglichst hohe Präzision erzielt? Dazu wurden zunächst die Abkürzungen aus den verschiedenen Beispielquelltexten aufgelöst, indem entweder das im Voraus trainierte „Word Embeddings for the Software Engineering Domain“ Modell [ECS18], kurz *SO_200*, oder ein eigens auf dem Quelltext selbst trainiertes *word2vec* Modell verwendet wurde. Das Ergebnis in Tabelle 9.13 zeigt, dass die Nutzung von *SO_200* allgemein bessere Ergebnisse hervorbringt, allerdings ist der Unterschied nur auf dem von Alatawi et al. bereitgestellten Beispielquelltext nennenswert.

Um auch den in Abschnitt 6.2.1 beschriebenen Vorschlag zu testen, wurde außerdem ein *word2vec*-Modell auf dem Quelltext von acht verschiedenen Programmen zur Steuerung von Drohnen produziert. Zudem wurde ein weiteres *word2vec*-Modell, beispielhaft für Domänenliteratur, auf dem englischsprachigen Wikipedia-Artikel über unbemannte Flugobjekte [Wik21b] trainiert. Mittels dieser Modelle wurde das Verfahren erneut mehrfach auf dem Quelltext von *Dronology* ausgeführt. Die Ergebnisse in Tabelle 9.14 zeigen erneut eine, wenn auch nicht allzu deutliche, Tendenz zum *SO_200*-Modell.

9.5.4 Reine Nutzung der Worteinbettung für Auswahl des Auflösungskandidaten

Um die auf Worteinbettungen basierenden Auflösungsverfahren direkt mit der präzisesten der Verfahren von Alatawi et al., als Vertreter des aktuellen Stands der Forschung, zu vergleichen, behandelt dieser Abschnitt **Forschungsfrage 6:** Wie präzise ist die ausschließliche Nutzung der Worteinbettungs-Ähnlichkeitsfunktion für die Auswahl des besten Auflösungskandidaten im Vergleich zu den Verfahren von Alatawi et al.? Dazu zeigt Tabelle 9.15 einen direkten Vergleich der beiden Verfahren auf verschiedenen Beispielquelltexten.

Tabelle 9.15: *word2vec*-basiertes Verfahren, mit *Kombinierte-Vektoren*-Ähnlichkeitsfunktion und *SO_200*-Worteinbettungsmodell, im Vergleich mit dem Bigramm-basierten Verfahren von Alatawi et al., unter Nutzung der Software-basierten Uni- und Bigramm-Grammatiken.

Quelltext	Verfahren	Präzision (Bezeichner)	Präzision (Token)
Alatawi et al.	*word2vec*	0,1538	0,3394
Alatawi et al.	Bigramm	0,6703	0,7828
Dronology	*word2vec*	0,0511	0,0651
Dronology	Bigramm	0,0511	0,0671
a2ps	*word2vec*	0,3629	0,2470
a2ps	Bigramm	0,3678	0,2544
which	*word2vec*	0,3952	0,2675
which	Bigramm	0,3938	0,2657

Auf fast allen Beispielprojekten unterliegt das *word2vec*-basierte Auflösungsverfahren dem Bigramm-basierten Ansatz von Alatawi et al. Auf dem von Alatawi et al. bereitgestellten Beispielquelltext ist dieser Unterschied deutlich, mit 51,65 Prozentpunkten, bezogen auf ganze Bezeichner. Bei den anderen Beispielquelltexten sind die beiden Verfahren vergleichbar. Nur auf *which* schlägt das worteinbettungsbasierte Verfahren das Bigramm-basierte um 0,14 Prozentpunkte, beziehungsweise um genau vier richtig aufgelöste Bezeichner-Vorkommnisse, von 2.242 gefundenen Vorkommnissen von 299 Bezeichnern aus der Musterlösung. Dieser Unterschied ist also nicht deutlich. Insgesamt lässt sich sagen, dass Worteinbettungen in dieser Verwendung, zumindest unter Nutzung dieses Modells, kein Potential im Vergleich zum aktuellen Stand der Forschung bieten. Da es sich bei *SO_200* [ECS18] um ein auf Software-bezogenen Datensätzen trainiertes Modell handelt, was, laut der Argumentation in Abschnitt 5.4.1.1 und mit den Verfahren von Alatawi et al. als Vorbild, für die geeignetere Wissensquelle gegenüber allgemeineren Sprachbibliotheken erachtet wird, ist die Wahl des Worteinbettungsmodells vermutlich nicht der Grund für die niedrige Präzision des Verfahrens.

9.5.5 Mehrdeutigkeitsauflösung bei mehreren gleichwertig besten Kandidaten

In Abschnitt 6.2.2 wird darauf eingegangen, dass die Verfahren von Alatawi et al. [AXX17] [AXY18], im Falle mehrerer gleichwertig besten Kandidaten, einen willkürlich auswählen.

Bei Betrachtung aller Vorkommnisse der 80 verschiedenen abgekürzter Bezeichner aus der Musterlösung für den Beispielquelltext Alatawi et al., haben von 659 Vorkommnissen 68 mehrere gleichwertig beste Auflösungskandidaten, und zwar bis zu 76 verschiedene. Im Schnitt haben die Vorkommnisse der abgekürzten Bezeichner aus der Musterlösung 2,73 verschiedene gleichwertig beste Auflösungskandidaten. Beschränkt auf die Vorkommnisse abgekürzter Bezeichner, welche tatsächlich mehrere gleichwertig beste Auflösungskandidaten haben, sind es im Schnitt 16,69. Der Bezeichner mit 76 verschiedenen gleichwertig besten ist „it“, und einige seiner Auflösungskandidaten werden in Beispiel C.3 aufgelistet.

Bezogen auf alle Quelltextbezeichner aus dem Beispielquelltext von Alatawi et al., bei denen von 16.565 Bezeichnern, mit im Schnitt 3,92 verschiedenen gleichwertig besten Auflösungskandidaten, 2.651 Bezeichner mehrere gleichwertig beste Auflösungskandidaten haben, und zwar im Schnitt 17,37 verschiedene, ist der Anteil und die Qualität an Bezeichnern mit mehreren Auflösungskandidaten im Schnitt durchaus vergleichbar. Allerdings hat aus dieser Menge der Bezeichner mit den meisten verschiedenen Auflösungskandidaten,

Tabelle 9.16: *word2vec*-basiertes Verfahren, unter Nutzung der *Kombinierte-Vektoren*-Ähnlichkeitsfunktion und des *SO_200*-Worteinbettungsmodells, zur Mehrdeutigkeitsauflösung des Unigramm-basierten Verfahrens von Alatawi et al., unter Nutzung der Software-basierten Unigramm-Grammatik, im Vergleich zum gleichen Verfahren mit willkürlicher Mehrdeutigkeitsauflösung.

Quelltext	Mehrdeutigkeitsauflösung	Präzision (Bezeichner)	Präzision (Token)
Alatawi et al.	*word2vec*	0,5824	0,7330
Alatawi et al.	willkürlich	0,5934	0,7376
Dronology	*word2vec*	0,0291	0,0535
Dronology	willkürlich	0,0511	0,0671
a2ps	*word2vec*	0,3678	0,2561
a2ps	willkürlich	0,3678	0,2556
which	*word2vec*	0,3943	0,2687
which	willkürlich	0,3938	0,2684

nämlich „set“, 692 verschiedenen Auflösungskandidaten. Einige dieser gleichwertig besten Auflösungskandidaten werden in Beispiel C.4 aufgelistet.

Dieser willkürlichen Auswahl unter diesen Kandidaten wird in Tabelle 9.16 eine Auswahl mittels der Ähnlichkeitsfunktion des *word2vec*-basierten Abkürzungsauflösungsverfahrens entgegengestellt. Dieser Vergleich liefert eine Antwort auf **Forschungsfrage 7:** Wie präzise ist die Mehrdeutigkeitsauflösung nach Auflösung mit den Verfahren von Alatawi et al., mittels worteinbettungsbasierter Entscheidung, im Vergleich zur willkürlichen Auswahl eines der gleichwertig besten Kandidaten? Dieses Auswahlverfahren hat keinen Vorteil gegenüber der willkürlichen Auswahl. In einigen Fällen ist das Ergebnis etwas besser, in einigen etwas schlechter. Der Unterschied ist aber nicht nennenswert.

10 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurden verschiedene Verfahren zur automatischen Auflösung von Abkürzungen in Quelltextbezeichnern vorgestellt. Diese wurden im Rahmen des *INDIRECT*-Projektes entwickelt und nutzen die von *INDIRECT* bereitgestellten Schnittstellen zur Erzeugung eines Absichtsmodells des Quelltextes und zur Implementierung darauf zugreifender Agenten. Um die vorgestellten Verfahren zu entwickeln wurden allgemeine Eigenschaften von Quelltext und darin enthaltenen Abkürzungen, sowie existierende Lösungen zur Auflösung von Abkürzungen in Quelltext analysiert, und andere Methoden der maschinellen Sprachverarbeitung und Informationsrückgewinnung vorgestellt, welche alternativ oder ergänzend zu den existierenden Lösungen eingesetzt werden könnten. Basierend auf diesen Beobachtungen wurden Verfahren auf Basis der Verfahren von Alatawi et al. [AXX17] [AXY18] entwickelt. Die Verfahren von Alatawi et al. verwenden unter anderem Unigramm- und Bigramm-Grammatiken zur Auswahl des besten Auflösungskandidaten, indem mit diesen Grammatiken die Wahrscheinlichkeit des Auftretens eines Auflösungskandidaten oder einer Abkürzung abgeschätzt wird. Diese Grammatiken werden in einem der hier vorgestellten Verfahren durch Trigramm-Grammatiken ergänzt. Weitere Verfahren verwenden verschiedene worteinbettungsbasierte Ähnlichkeitsfunktionen, um den Schritt der Entscheidung des besten Auflösungskandidaten von Alatawi et al. zu ersetzen oder zu ergänzen. Bei diesen Verfahren wurde auch, wie bei Alatawi et al., die Entscheidung der Spaltung eines Bezeichners implizit nach Entscheidung der Auflösung getroffen. Während Alatawi et al. nur jene abgekürzten Bezeichner auflösen, welche in ihrer eigens erzeugten Musterlösung vorkommen, lösen die hier vorgestellten Verfahren alle Bezeichner aus dem Quelltext auf.

Nur eines der entwickelten Verfahren, die Trigramm-basierte Erweiterung der Verfahren von Alatawi et al., bot einen leichten Vorteil gegenüber einer imitierten Implementierung des besten Verfahrens von Alatawi et al. [AXY18]. Konkret erzielt das Trigramm-basierte Verfahren, unter Einsatz der Software-basierten, von Alatawi et al. bereitgestellten Unigramm- und Bigramm-Grammatiken und einer aus dem Google Books N-Gramm Korpus [Goo21b] stammenden Trigramm-Grammatik, auf einem von Alatawi et al. bereitgestelltem Beispielquelltext, eine Präzision von 70,33% richtig aufgelösten abgekürzten Bezeichnern. Das nachimplementierte, Bigramm-basierte Verfahren von Alatawi et al. erzielt, ebenfalls unter Verwendung der Software-basierten Uni- und Bigramme, auf dem gleichen Beispielquelltext, eine Präzision von 67,03%. Die externe Gültigkeit dieses entdeckten Vorteils ist allerdings in Frage zu stellen, da das imitierte Verfahren, welches für den Vergleich benutzt wurde, nicht an die Qualität des Originals herankommt. Dieses

wurde allerdings auch als Grundlage für das Trigramm-basierte Verfahren verwendet, der intern beobachtete Vorteil könnte also dennoch verallgemeinerbar sein.

Zukünftige Arbeiten im Gebiet der Auflösung von Abkürzungen in Quelltext könnten auf dieser Beobachtung aufbauen, und das Trigramm-basierte Verfahren optimieren, aber auch weitere N-Gramm-basierte Verfahren darauf aufbauen. Obwohl die hier vorgestellte Verwendung von Worteinbettungen keine guten Ergebnisse geliefert hat, könnten zudem andere Methoden entwickelt werden, um Worteinbettungen zur Abkürzungsauflösung einzusetzen. Beispielsweise könnten Ähnlichkeitsfunktionen für Worteinbettungen nicht als alleiniges Mittel zur Bewertung einer Auflösung verwendet werden, sondern im Ansatz von Alatawi et al. als Ersatz für die N-Gramm-Grammatiken zur Bewertung der Auftretenswahrscheinlichkeit eines Kandidaten oder einer Abkürzung verwendet werden. Außerdem könnten beispielsweise Methoden, welche auf bestimmte Abkürzungstypen oder Kontexte spezialisiert sind, zu einem alle Kontexte und Abkürzungstypen umfassenden Verfahren kombiniert, oder ergänzend zu einem allgemeineren Verfahren genutzt werden. Auch einige der Methoden, welche hier oder in anderen verwandten Arbeiten eingesetzt wurden, so wie weitere Methoden der maschinellen Sprachverarbeitung, könnten zum gegenseitigen Vorteil kombiniert werden. Für die Kandidatensuche im Quelltext verwenden Alatawi et al. ein naives Spaltverfahren, welches durch ein besseres, wie *TRIS* [GGG+12], *LINSEN* [CDMM12] oder *GenTest* [LBM10] ersetzt werden könnte. Wortartenmarkierer, welche hier mit wenig Erfolg in einem der worteinbettungsbasierten Verfahren eingesetzt wurden, könnten in anderer Verwendung weiterhin Vorteile bieten. Beispielsweise könnten die Wortarten verwendet Werden, um den grammatischen Zusammenhang der Token eines zusammengesetzten Begriffes zu erkennen, und somit eine Aussage über die Wahrscheinlichkeit eines Auflösungskandidaten über dessen Grammatik getroffen werden. Eine einheitliche Stammwortzerlegung oder Normalisierung der Auflösungskandidaten und der Auflösungen in der Musterlösung könnte vermeiden, dass Auflösungen, welche für die Informationsrückgewinnung ausreichend, aber nicht mit der in der Musterlösung vermerkten Auflösung identisch sind, als falsch angesehen werden. Die Verwendung einer solchen Normalisierung würde allerdings eine Umdeutung des Begriffs der richtigen Auflösung erfordern.

Literaturverzeichnis

[a2p21] *a2ps-4.14*. Version: 2021. http://www.linuxfromscratch.org/blfs/view/8.3/pst/a2ps.html (zitiert auf Seite 78).

[APS16] Arntz, Reiner ; Picht, Heribert ; Schmitz, Klaus-Dirk: *Einführung in die Terminologiearbeit*. Georg Olms e.K. Verlagsbuchhandlung Hagentorwall 7 31134 Hildesheim : Georg Olms Verlag, 2016. – ISBN 978–3–487–15056–7. – Google-Books-ID: 6ma_DQAAQBAJ (zitiert auf Seite 4).

[AXX17] Alatawi, Abdulrahman ; Xu, Weifeng ; Xu, Dianxiang: Bayesian Unigram-Based Inference for Expanding Abbreviations in Source Code. In: *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, 2017, S. 543–550. – ISSN: 2375-0197 (zitiert auf den Seiten 16, 18, 29, 39, 45, 46, 48, 49, 51, 55, 57, 60, 62, 65, 75, 87 und 89).

[AXY18] Alatawi, Abdulrahman ; Xu, Weifeng ; Yan, Jie: The Expansion of Source Code Abbreviations Using a Language Model. In: *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)* Bd. 02, 2018, S. 370–375. – ISSN: 0730-3157 (zitiert auf den Seiten 17, 20, 39, 45, 46, 48, 49, 55, 57, 60, 62, 63, 65, 68, 75, 82, 87 und 89).

[BB79] Bouwhuis, Don ; Bouma, Herman: Visual word recognition of three-letter words as derived from the recognition of the constituent letters. 25 (1979), Nr. 1, 12–22. http://dx.doi.org/10.3758/BF03206104. – DOI 10.3758/BF03206104. – ISSN 1532–5962 (zitiert auf Seite 32).

[BDLM09] Binkley, Dave ; Davis, Marcia ; Lawrie, Dawn ; Morrell, Christopher: To camelcase or under_score. In: *2009 IEEE 17th International Conference on Program Comprehension*, 2009, S. 158–167. – ISSN: 1092-8138 (zitiert auf Seite 6).

[BDVJ03] Bengio, Yoshua ; Ducharme, Réjean ; Vincent, Pascal ; Jauvin, Christian: A Neural Probabilistic Language Model. (2003), S. 19 (zitiert auf Seite 6).

[BF06] Brants, Thorsten ; Franz, Alex: *LD Consortium*. http://dx.doi.org/10.35111/CQPA-A498. Version: 2006. – type: dataset (zitiert auf den Seiten 17, 18, 45, 50 und 68).

[BGJM17] Bojanowski, Piotr ; Grave, Edouard ; Joulin, Armand ; Mikolov, Tomas: Enriching Word Vectors with Subword Information. (2017). http://arxiv.org/abs/1607.04606 (zitiert auf Seite 52).

[BYP96] Baeza-Yates, Ricardo A. ; Perleberg, Chris H.: Fast and practical approximate string matching. (1996), S. 7 (zitiert auf Seite 14).

[CAHV15] Carvalho, Nuno R. ; Almeida, José J. ; Henriques, Pedro R. ; Varanda, Maria J.: From source code identifiers to natural language terms. 100

(2015), 117–128. http://dx.doi.org/10.1016/j.jss.2014.10.013. – DOI 10.1016/j.jss.2014.10.013. – ISSN 0164–1212 (zitiert auf Seite 20).

[CDMM12] Corazza, Anna ; Di Martino, Sergio ; Maggio, Valerio: LINSEN: An efficient approach to split identifiers and expand abbreviations. In: *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, 2012, S. 233–242. – ISSN: 1063-6773 (zitiert auf den Seiten 14, 18, 19, 29, 30, 32, 36, 38, 45, 46, 65 und 90).

[Cra21] Crawl, Common: *Common Crawl*. Version: 2021. https://commoncrawl.org/ (zitiert auf den Seiten 69 und 85).

[DIN11] Norm DIN 2342 August 2011. *Begriffe der Terminologielehre* (zitiert auf Seite 4).

[DJ08] Daniel Jurafsky, James H. M.: *Speech and Language Processing - Jurafsky - Second Edition*. Second. Upper Saddle River, New Jersey 07458 : Pearson Prentice Hall, 2008. – ISBN 978–0–13–504196–3 (zitiert auf den Seiten 3 und 5).

[DK00] Deemter, Kees v. ; Kibble, Rodger: On Coreferring: Coreference in MUC and Related Annotation Schemes. 26 (2000), Nr. 4, 629–637. http://dx.doi.org/10.1162/089120100750105966. – DOI 10.1162/089120100750105966. – ISSN 0891–2017, 1530–9312 (zitiert auf Seite 5).

[ECS18] Efstathiou, V. ; Chatzilenas, C. ; Spinellis, D.: Word Embeddings for the Software Engineering Domain. In: *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, 2018, S. 38–41. – ISSN: 2574-3864 (zitiert auf den Seiten 68, 83, 84, 86 und 87).

[EHPVS09] Enslen, Eric ; Hill, Emily ; Pollock, Lori ; Vijay-Shanker, K.: Mining source code to automatically split identifiers for software analysis. In: *2009 6th IEEE International Working Conference on Mining Software Repositories*, 2009, S. 71–80. – ISSN: 2160-1860 (zitiert auf den Seiten 21 und 38).

[eL09] Řehůřek, Radim ; Ltd., RARE T.: *Gensim: topic modelling for humans*. Version: 2009. https://radimrehurek.com/gensim/ (zitiert auf Seite 69).

[Eur20] Eurich, Felix: *Entwurf und Aufbau einer semantischen Repräsentation von Quelltext*, Karlsruher Institut für Technologie (KIT) – IPD Tichy, Master's Thesis, january 2020. https://code.ipd.kit.edu/hey/indirect/wikis/Theses/eurich_ma (zitiert auf den Seiten 61 und 64).

[Fac21] Facebook: *English word vectors · fastText*. Version: 2021. https://fasttext.cc/index.html (zitiert auf den Seiten 69 und 85).

[FBC01] Feng, Fangfang ; Bruce Croft, W.: Probabilistic techniques for phrase extraction. 37 (2001), Nr. 2, 199–220. http://dx.doi.org/10.1016/S0306-4573(00)00029-7. – DOI 10.1016/S0306–4573(00)00029–7. – ISSN 03064573 (zitiert auf Seite 11).

[FBL06] Feild, Henry ; Binkley, David ; Lawrie, Dawn: AN EMPIRICAL COMPARISON OF TECHNIQUES FOR EXTRACTING CONCEPT ABBREVIATIONS FROM IDENTIFIERS. (2006), S. 6 (zitiert auf den Seiten 20 und 38).

[FK79] Francis, W. N. ; Kucera, H.: Brown Corpus Manual. (1979). https://www.cis.uni-muenchen.de/~micha/kurse/lg-SS2009/Brown_Corpus_Manual.html (zitiert auf Seite 22).

[GGG+12] Guerrouj, L. ; Galinier, P. ; Guéhéneuc, Y. ; Antoniol, G. ; Penta, M. D.: TRIS: A Fast and Accurate Identifiers Splitting and Expansion Algorithm. In: *2012 19th Working Conference on Reverse Engineering*, 2012, S. 103–112. – ISSN: 2375-5369 (zitiert auf den Seiten 16, 20, 29, 32, 36, 39, 46, 55 und 90).

[Git21] *Github*. Version: 2021. https://github.com (zitiert auf Seite 68).

[Goo21a] Google: *Google C++ Style Guide*. Version: 2021. https://google.github.io/styleguide/cppguide.html (zitiert auf Seite 37).

[Goo21b] Google: *Google Ngram Viewer*. Version: 2021. http://storage.googleapis.com/books/ngrams/books/datasetsv2.html (zitiert auf den Seiten 68 und 89).

[GPAG13] Guerrouj, Latifa ; Penta, Massimiliano D. ; Antoniol, Giuliano ; Guéhéneuc, Yann-Gaël: TIDIER: an identifier splitting approach using speech recognition techniques. 25 (2013), Nr. 6, 575–599. http://dx.doi.org/10.1002/smr.539. – DOI 10.1002/smr.539. – ISSN 2047–7481. – _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.539 (zitiert auf Seite 20).

[Har54] Harris, Zellig S.: Distributional Structure. 10 (1954), Nr. 2, 146–162. http://dx.doi.org/10.1080/00437956.1954.11659520. – DOI 10.1080/00437956.1954.11659520. – ISSN 0043–7956, 2373–5112 (zitiert auf Seite 6).

[Hey19] Hey, T.: INDIRECT: Intent-Driven Requirements-to-Code Traceability. In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2019, S. 190–191. – ISSN: 2574-1934 (zitiert auf den Seiten 9, 62 und 75).

[HFB+08] Hill, Emily ; Fry, Zachary P. ; Boyd, Haley ; Sridhara, Giriprasad ; Novikova, Yana ; Pollock, Lori ; Vijay-Shanker, K.: AMAP: automatically mining abbreviation expansions in programs to enhance software maintenance tools. In: *Proceedings of the 2008 international working conference on Mining software repositories*. Newark, DE 19716 USA : Association for Computing Machinery, 2008 (MSR '08). – ISBN 978–1–60558–024–1, 79–88 (zitiert auf den Seiten 12, 19, 29, 32, 39, 45 und 46).

[Jen14] Jenks, Grant: wordsegment: English word segmentation. (2014). http://www.grantjenks.com/docs/wordsegment/ (zitiert auf den Seiten 17 und 45).

[JGDG10] Ji, Heng ; Grishman, Ralph ; Dang, Hoa T. ; Griffitt, Kira: Overview of the TAC2010 Knowledge Base Population (KBP) Track. (2010), S. 56 (zitiert auf Seite 23).

[JLZZ18] Jiang, Yanjie ; Liu, Hui ; Zhu, Jia Q. ; Zhang, Lu: Automatic and Accurate Expansion of Abbreviations in Parameters. (2018), S. 1–1. http://dx.doi.org/10.1109/TSE.2018.2868762. – DOI 10.1109/TSE.2018.2868762. – ISSN 1939–3520. – Conference Name: IEEE Transactions on Software Engineering (zitiert auf den Seiten 18, 20, 29, 36, 42, 45, 46, 48 und 51).

[jna21] *java-native-access/jna*. Version: 2021. https://github.com/java-native-access/jna. – original-date: 2011-04-29T13:30:29Z (zitiert auf Seite 64).

[LB11] Lawrie, Dawn ; Binkley, Dave: Expanding identifiers to normalize source code vocabulary. In: *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, 2011, S. 113–122. – ISSN: 1063-6773 (zitiert auf den Seiten 13, 15, 16, 19, 36, 44, 65 und 67).

[LBM10] Lawrie, Dawn ; Binkley, David ; Morrell, Christopher: Normalizing Source Code Vocabulary. In: *Working Conference on Reverse Engineering, WCRE*, 2010, S. 3–12 (zitiert auf den Seiten 13, 15, 32, 38, 39 und 90).

[LFB07] Lawrie, Dawn ; Feild, Henry ; Binkley, David: Extracting Meaning from Abbreviated Identifiers. In: *Seventh IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2007)*, 2007, S. 213–222 (zitiert auf den Seiten 11, 15, 19, 36, 38, 44 und 48).

[LGM+15] Liu, Yue ; Ge, Tao ; Mathews, Kusum S. ; Ji, Heng ; McGuinness, Deborah L.: Exploiting Task-Oriented Resources to Learn Word Embeddings for Clinical Abbreviation Expansion. (2015), 92–97. http://dx.doi.org/10.18653/v1/W15-3810. – DOI 10.18653/v1/W15–3810 (zitiert auf den Seiten 23, 36, 47 und 51).

[LML+20] Lin, Yuri ; Michel, Jean-Baptiste ; Lieberman, Erez A. ; Orwant, Jon ; Brockman, Will ; Petrov, Slav: Syntactic Annotations for the Google Books NGram Corpus. (20120), S. 6 (zitiert auf Seite 82).

[MCCD13] Mikolov, Tomas ; Chen, Kai ; Corrado, Greg ; Dean, Jeffrey: Efficient Estimation of Word Representations in Vector Space. (2013). http://arxiv.org/abs/1301.3781 (zitiert auf den Seiten 23 und 52).

[MGDP+10] Madani, Nioosha ; Guerrouj, Latifa ; Di Penta, Massimiliano ; Guéhéneuc, Yann-Gaël ; Antoniol, Giuliano: Recognizing Words from Source Code Identifiers Using Speech Recognition Techniques. In: *Recognizing Words from Source Code Identifiers Using Speech Recognition Techniques*, 2010, S. 68–77 (zitiert auf den Seiten 15 und 32).

[MKB+10] Mikolov, Tomas ; Karafiat, Martin ; Burget, Lukas ; Cernocky, Jan ; Khudanpur, Sanjeev: Recurrent Neural Network Based Language Model. (2010), S. 4 (zitiert auf Seite 6).

[MRS18] Manning, Christopher D. ; Raghavan, Prabhakar ; Schütze, Hinrich: Introduction to Information Retrieval. (2018), S. 569 (zitiert auf den Seiten 3, 4 und 5).

[MSB+14] Manning, Christopher ; Surdeanu, Mihai ; Bauer, John ; Finkel, Jenny ; Bethard, Steven ; McClosky, David: The Stanford CoreNLP Natural Language Processing Toolkit. In: *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, Association for Computational Linguistics, 2014, 55–60 (zitiert auf Seite 58).

[MYZ13] Mikolov, Tomas ; Yih, Wen-tau ; Zweig, Geoffrey: Linguistic Regularities in Continuous Space Word Representations. (2013), S. 6 (zitiert auf den Seiten xiii, 6 und 7).

[ND21] Notre Dame, University of: *Dronology – A Software Engineering Research Environment using Unmanned Aerial Systems*. Version: 2021.

https://\protect\unhbox\voidb@x\bgroup\def.{Dronology}\let\
futurelet\@let@token\let\itshapeDronology\egroup.info/ (zitiert auf den Seiten 65, 69 und 78).

[NDA+19] Newman, Christian D. ; Decker, Michael J. ; Alsuhaibani, Reem S. ; Peruma, Anthony ; Kaushik, Dishant ; Hill, Emily: An Empirical Study of Abbreviations and Expansions in Software Artifacts. In: *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2019, S. 269–279. – ISSN: 2576-3148 (zitiert auf den Seiten xv, 19, 28, 29, 31, 32, 33, 34, 40, 41, 42, 46 und 48).

[NLT21] NLTK: *Natural Language Toolkit — NLTK 3.5 documentation*. Version: 2021. https://www.nltk.org/ (zitiert auf Seite 68).

[OHI21] Owden, David ; Hamish ; Ingram, Jonathan: *sajari/word2vec*. Version: 2021. https://github.com/sajari/word2vec. – original-date: 2015-08-04T21:59:57Z (zitiert auf Seite 64).

[Ora21] Oracle: *Code Conventions for the Java Programming Language: 9. Naming Conventions*. Version: 2021. https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html (zitiert auf Seite 37).

[Par21] Parr, Terence: *ANTLR*. Version: 2021. https://www.antlr.org/ (zitiert auf Seite 61).

[PSM14] Pennington, Jeffrey ; Socher, Richard ; Manning, Christopher: Glove: Global Vectors for Word Representation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Association for Computational Linguistics, 2014, 1532–1543 (zitiert auf Seite 52).

[Rad15] Radyushin, Alex: 848 000 Medical Acronyms and Medical Abbreviations. (2015). https://www.allacronyms.com/_medical (zitiert auf Seite 23).

[RG04] Ratinov, L. ; Gudes, E.: Abbreviation Expansion in Schema Matching and Web Integration. In: *IEEE/WIC/ACM International Conference on Web Intelligence (WI'04)*, 2004, S. 485–489 (zitiert auf den Seiten 21, 47 und 48).

[RWC21] Rossum, Guido van ; Warsaw, Barry ; Coghlan, Nick: *PEP 8 – Style Guide for Python Code*. Version: 2021. https://www.python.org/dev/peps/pep-0008/ (zitiert auf Seite 37).

[Sin01] Singhal, Amit: Modern Information Retrieval: A Brief Overview. (2001), S. 9 (zitiert auf Seite 7).

[Sta21] *Stack Overflow - Where Developers Learn, Share, & Build Careers*. Version: 2021. https://stackoverflow.com/ (zitiert auf Seite 68).

[Web21] *Webster's Unabridged Dictionary*. Version: 2021. https://www.gutenberg.org/ebooks/29765 (zitiert auf den Seiten 68 und 80).

[Weg18] Wegner, Peter: Concepts and paradigms of object-oriented programming. 1 (2018), Nr. 1. https://dl.acm.org/doi/pdf/10.1145/382192.383004. – Brown University (zitiert auf Seite 6).

[Wik21a] *Konsonantenschrift*. Version: 2021. https://de.wikipedia.org/w/index.php?title=Konsonantenschrift&oldid=201127174. – Page Version ID: 201127174 (zitiert auf Seite 32).

[Wik21b] *Unmanned aerial vehicle*. Version: 2021. https://en.wikipedia.org/w/index.php?title=Unmanned_aerial_vehicle&oldid=1013191516. – Page Version ID: 1013191516 (zitiert auf den Seiten 69 und 86).

[Wik21c] *SMS language*. Version: 2021. https://en.wikipedia.org/w/index.php?title=SMS_language&oldid=1005039993. – Page Version ID: 1005039993 (zitiert auf Seite 32).

[Woo21] Wood, Carlo: *GnuWin - Browse /which/2.20 at SourceForge.net*. Version: 2021. https://sourceforge.net/projects/gnuwin32/files/which/2.20/ (zitiert auf Seite 78).

[WT15] Weigelt, Sebastian ; Tichy, Walter: Poster: ProNat: An Agent-Based System Design for Programming in Spoken Natural Language. (2015), 05, S. 819–820. http://dx.doi.org/10.1109/ICSE.2015.264. – DOI 10.1109/ICSE.2015.264 (zitiert auf Seite 9).

[XXA+18] Xu, Weifeng ; Xu, Dianxiang ; Alatawi, Abdulrahman ; Ariss, Omar E. ; Liu, Yunkai: Statistical Unigram Analysis for Source Code Repository. (2018). http://dx.doi.org/10.1142/S1793351X18400123. – DOI 10.1142/S1793351X18400123. – Publisher: World Scientific Publishing Company (zitiert auf den Seiten 17, 18, 30, 45, 49, 50, 68 und 82).

[ZSST11] Zhang, Wei ; Sim, Yan-Chuan ; Su, Jian ; Tan, Chew-Lim: Entity Linking with Effective Acronym Expansion, Instance Selection and Topic Modeling. In: *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011, 1909–1914 (zitiert auf den Seiten 22, 36 und 47).

Anhang

A Anhang: Entwurf

Tabelle A.1 zeigt eine Liste der Programme, auf deren Quelltext, wie in Abschnitt 6.2.1 beschrieben, ein Worteinbettungsmodell trainiert wurde, um Abkürzungen in *Dronology* aufzulösen. Dies dient in Abschnitt 9.5.3 der Beantwortung der **Forschungsfrage 5**, auf welchen Wissensquellen ein Worteinbettungsmodell trainiert werden sollte, damit ein darauf aufbauendes Abkürzungsauflösungsverfahren eine möglichst hohe Präzision erzielt.

B Anhang: Datensätze

B.1 Abkürzungen aus Musterlösung von Alatwi et al., welche nicht im jeweiligen Beispielquelltext auftreten

Diese Abkürzungen, aus der durch Alatawi et al., für deren eigenen Beispielquelltext, zur Verfügung gestellten Musterlösung, treten in eben diesem Beispielquelltext nicht, oder nicht als eigenständige Bezeichner, auf.

- „ta“ für „text annotation“ ist nur im Quelltextbezeichner „xyta“ zu finden.
- „uisp“ für „user interface setting parameters“ ist nur im Quelltextbezeichner „updateUISP“ zu finden.

Tabelle A.1: Dronen-Steuerungs-Programme, welche für das Training eines Worteinbettungsmodells verwendet wurden.

Name	Website
PX4-Autopilot	https://px4.io/
Paparazzi	http://paparazziuav.org/
ArduPilot	http://ardupilot.org/
LibrePilot	https://librepilot.org/
Flone	https://flone.cc/
ODM	https://opendronemap.org/
DronePan	http://dronepan.com/
Dronology	https://*Dronology*.info/

Alle Quelltexte wurden am 17. März 2021 heruntergeladen.

- „ js“ für „ joined subclass“ ist nicht im Quelltext zu finden, die Musterlösung deutet aber auf Quelltextzeile mit dem Bezeichner „JoinedSubclass“.
- „tinum“ für „task id number“ ist nicht im Quelltext zu finden, die Musterlösung deutet eine Quelltextzeile entfernt von Kommentar mit der richtigen Auflösung.
- „fa“ für „formated array“ ist nicht im Quelltext zu finden, die Musterlösung deutet aber auf eine Quelltextzeile mit dem Bezeichner „FormatterArray“.

B.2 Abgekürzte Bezeichner aus der Musterlösung für *Dronology*, welche nur vier Quelltextdateien abdeckt

Die in Abschnitt 8.2 beschriebene Musterlösung, welche für vier Quelltextdateien von *Dronology* angelegt wurde, und die Quelltextdatei und -Zeile des Vorkommnisses eines Bezeichners berücksichtigt, beinhaltet 22 verschiedene abgekürzte Bezeichner:

- „msg“ für „message“
- „util“ für „utility“
- „iFlightDirector“ für „interface Flight Director“
- „iDrone“ für „interface Drone“
- „xml“ für „extensible markup language“
- „LoadXMLFlight“ für „Load extensible markup language Flight“
- „int“ für „integer“
- „safetyMgr“ für „safety Manager“
- „fzView“ für „flight zone View“
- „baseMgr“ für „base Manager“
- „println“ für „print line“
- „ getFlightID“ für „ get Flight Identifier“
- „e“ für „error“
- „e1“ für „error 1“
- „maxAltitude“ für „maximum Altitude“
- „maxAlt“ für „maximum Altitude“
- „abs“ für „absolute“
- „alt“ für „altitude“
- „lon“ für „longitude“
- „lat“ für „lattitude“
- „o“ für „object“
- „coord“ für „coordinates“

C Anhang: Evaluation

C.1 Zusammengesetzte Bezeichner aus *Dronology*, welche nicht gespalten oder aufgelöst wurden

In Abschnitt 9.3 wird festgestellt, dass das Unigramm-basierte Auflösungsverfahren keinen einzigen zusammengesetzten Bezeichner aus den vier betrachteten Quelltextdateien von *Dronology* richtig spaltet, geschweige denn auflöst. Einige Beispiele dafür befinden sich in Beispiel C.1.

Beispiel C.1: Zusammengesetzte Bezeichner, welche in Abschnitt 9.3 nicht richtig gespalten oder aufgelöst wurden.

- Der Bezeichner „setAltitude“ wurde zu „set typecast latitude“ aufgelöst, anstatt ihn zu „set Altitude“ zu spalten.
- Der Bezeichner „setLongitude“ wurde nicht verändert, anstatt ihn zu „set Longitude“ zu spalten.
- Der Bezeichner „toString“ wurde nicht verändert, anstatt ihn zu „to String“ zu spalten.
- Der Bezeichner „iFlightDirector“ wurde nicht verändert, anstatt ihn zu „interface Flight Director“ aufzulösen.
- Der Bezeichner „xml“ wurde zu „xmlf model light“ aufgelöst, anstatt ihn zu „extensible markup language“ aufzulösen.
- Der Bezeichner „safetyMgr“ wurde nicht verändert, anstatt ihn zu „safety Manager“ aufzulösen.
- Der Bezeichner „fzView“ wurde nicht verändert, anstatt ihn zu „flight zone View“ aufzulösen.

C.2 Abgekürzte Bezeichner aus *Dronology*, welche durch das Bigramm-basierte Verfahren richtig aufgelöst wurden

Folgendes ist eine Liste aller abgekürzten Bezeichner aus *Dronology*, welche in Abschnitt 9.2.3 durch die Imitation des Software-basierten Bigramm-Verfahrens von Alatawi et al. richtig aufgelöst wurden:

- „alt“ für „altitude“
- „lat“ für „latitude“
- „lon“ für „longitude“
- „pnt“ für „point“
- „tgt“ für „target“
- „p“ für „point“
- „img“ für „image“
- „sim“ für „simulator“
- „t“ für „text“
- „fos“ für „file output stream“

- „bw“ für „buffered writer“
- „itr“ für „iterator“
- „btn“ für „button“
- „fzv“ für „flight zone view“
- „gc“ für „graphics context“
- „zb“ für „zone bounds“
- „maxAlt“ für „maximum Altitude“

C.3 Abgekürzte Bezeichner aus *Dronology*, welche durch das Bigramm-basierte Verfahren falsch aufgelöst wurden.

Beispiel C.2 ist eine Liste an Beispielen abgekürzten Bezeichner aus *Dronology*, welche in Abschnitt 9.2.3 durch die Imitation des Software-basierten Bigramm-Verfahrens von Alatawi et al. falsch aufgelöst wurden.

Beispiel C.2: Abgekürzte Bezeichner aus *Dronology*, welche durch die Imitation des Bigramm-basierten Verfahrens von Alatawi et al., unter Nutzung der Software-basierten Grammatiken, falsch aufgelöst wurden.

- „safetyMgr“ für „safety Manager“
- „fzView“ für „flight zone View“
- „e“ für „exception“
- „xml“ für „extensible markup language“
- „LoadXMLFlight“ für „Load Extensible Markup Language Flight“
- „westLon“ für „west Longitude“
- „coord“ für „coordinates“
- „o“ für „object“
- „alt“ für „altitude“
- „lat“ für „latitude“
- „lon“ für „longitude“
- „sim“ für „simulator“
- „gc“ für „graphics context“

C.4 Mehrdeutigkeitsauflösung bei mehreren gleichwertig besten Auflösungskandidaten

In Abschnitt 6.2.2 wird darauf eingegangen, dass es in den Verfahren von Alatawi et al. mehrere gleichwertig beste Auflösungskandidaten geben kann, unter denen einer willkürlich ausgewählt wird. Der abgekürzte Bezeichner mit den meisten gleichwertig besten Auflösungskandidaten aus dem Beispielquelltext von Alatawi et al., welcher in der Musterlösung vorkommt, ist „it“, mit 76 Kandidaten. Einige dieser Kandidaten werden in Beispiel C.3 aufgelistet.

Beispiel C.3: Eine Liste einiger gleichwertig bester Auflösungskandidaten für den abgekürzten Quelltextbezeichner „it“.

- „id to“
- „id to“
- „iautocmt“
- „ict timestamp“
- „int“
- „int tx“
- „iautocmt tx“
- „identity true“
- „is time“
- „iuname transaction“

Der nicht abgekürzte Bezeichner mit den meisten gleichwertig besten Auflösungskandidaten ist „set“, mit 692 Kandidaten. Einige dieser Kandidaten werden in Beispiel C.4 aufgelistet.

Beispiel C.4: Eine Liste einiger gleichwertig bester Auflösungskandidaten für den Quelltextbezeichner „set“.

- „serializable early tracev“
- „session exit two“
- „status entry todo“
- „some entity“
- „string em trace“
- „starting early to“
- „saved entry“
- „saved enabled the“
- „session existing todo“
- „some existing type“