Entwicklung und Vergleich von Algorithmen zur Umformung von Neben- in Hauptsätze

Masterarbeit

von

Sebastian Bischoff

aus

Neuss

vorgelegt am

Lehrstuhl für Rechnernetze
Prof. Dr. Martin Mauve
Heinrich-Heine-Universität Düsseldorf

Mai 2021

Betreuer:
Markus Brenneis, M. Sc.

Zusammenfassung

In Dialog-basierten Onlineforen werden Argumente zwischen Teilnehmern ausgetauscht. D-BAS [Kra+18] unterstützt bei der Diskussion die Anwender, indem unter anderem ein Statement für den Benutzer herausgesucht wird, welches der Teilnehmer in Form eines Nebensatzes begründen kann. Diese Begründung wird zeitversetzt einem anderen Teilnehmer der Diskussion, zum Beispiel der Gegenseite, präsentiert, wodurch dieser dann auf diese Begründung in verschiedenen Formen eingehen kann. Dabei verändert sich die Satzstruktur der Begründung nicht. Das heißt, dass alle Benutzer die Begründungen und Argumente in Form eines Nebensatzes angezeigt bekommen. Während der Interaktion mit dem System der zeitversetzten Diskussion erweist sich dies als Hindernis für die Teilnehmer.

In dieser Arbeit stellen wir einen Algorithmus vor, welcher solche Nebensätze anhand von Werkzeugen aus dem Bereich der Neuro-Linguistischen Programmierung in Hauptsätze umwandelt. Wir implementieren mithilfe von POS-Tagging, Dependency-Parsing und handgeschriebenen Regeln einen solchen Algorithmus. Sowohl während der Implementierungsphase als auch bei der Auswertung des Algorithmus testen und werten wir diesen anhand von Daten aus zwei Feldexperimenten aus dem Dialog-basierten Onlineforum D-BAS [KMM17] aus. Wir stellen innerhalb der Implementierungsphase fest, dass es bei den Benutzereingaben und bei den Umwandlungen durch den Dependency-Parser zu einigen teils abhängigen und teils unabhängigen Fehlern kommt, welche manuell behoben werden müssen. Die Auswertung des Testsets zeigt, dass eine korrekte Transformationsrate von ungefähr 67 % erreicht wird. Ein Teil der nicht korrekt umgewandelten Datensätze können wegen der Komplexität der deutschen Sprache nur sehr schwer umgewandelt werden und demzufolge ist die Implementierung eines erfolgreichen Algorithmus keine leichte Aufgabe. Im Vergleich zur vorherigen Arbeit von Borodkin [Bor20] erreichen wir erhebliche Verbesserungen bezüglich der erfolgreichen Umwandlung von Neben- in Hauptsätzen. Die Verschlechterung des Verständnisses der Sätze bei einer falschen Umwandlung und die Schwierigkeit bei der automatischen Erkennung von Eingaben, welche nicht korrekt umgewandelt werden können, führen dazu, dass der Algorithmus dennoch in der aktuellen Form nicht für die Benutzung in D-BAS freigegeben werden sollte. Der vorgestellte primitive Ansatz zur Filterung der Benutzereingaben, welche der Algorithmus voraussichtlich korrekt umwandeln kann, scheint vielversprechend zu sein, bedarf aber weiterer Entwicklung. Wir glauben, dass zwar der Ansatz des Dependency-Parsings alleine nicht die gewünschte Umwandlung aller Sätze erzielen kann, aber mit der Verknüpfung anderer Methoden einen vielversprechenden Ansatz bietet.

Inhaltsverzeichnis

Abbildungsverzeichnis **v**

Tabellenverzeichnis **vi**

1 Einleitung **1**

1.1 Verwandte Arbeiten 3

1.2 Aufbau der Arbeit 4

2 Grundlagen **5**

2.1 Testsets und Trainingssets 5

2.2 Part-Of-Speech Tagging 6

2.3 Dependency Parser 7

3 Implementierung **9**

3.1 CoNLL-Format 10

3.2 Konvertierung des Nebensatzes in einen Hauptsatz 11

3.3 Abweichungen vom Idealfall und zusätzliche Sonderfälle 15

3.3.1 Allgemeine Problemstellung 15

3.3.2 Konkrete Sonderfälle 16

3.3.3 Falsch umgewandelte Sätze im Trainingsset 19

4 Evaluation **22**

4.1 Auswertungsmetrik 22

4.2 Auswertung des Testsets 24

4.2.1 Allgemeine Auswertung 24

4.2.2 Analyse richtiger Eingaben und falscher Transformationen 26

4.2.3 Analyse falscher Eingaben und richtiger Transformationen 30

4.2.4 Verbesserungen/Verschlechterungen des Verständnisses von Sätzen . 30

4.3 Vergleich zur anderen Arbeit 31
4.3.1 Anpassung der Bewertungsmetrik 32
4.3.2 Vergleich der subjektiven Bewertungen 34
4.3.3 Vergleich der Ergebnisse 35
4.4 Ansatz zum Filtern der Benutzereingaben 36
4.4.1 Implementierung des Filters 36
4.4.2 Analyse des Trainingssets 37
4.4.3 Analyse des Testsets 38

5 Fazit und Ausblick **44**

Literatur **47**

Abbildungsverzeichnis

1.1 Beispiel von einer Diskussion aus D-BAS 2

2.1 Beispielsatz mit zugehörigem Graphen 8

3.1 Überblick über den gesamten Ablauf der Transformation von Neben- in Hauptsätzen 11
3.2 Ursprünglicher Nebensatz wird während der Transformation in Haupt- und Nebensatz aufgeteilt 12
3.3 Programmablauf der Transformation vom CoNLL-Format zum Hauptsatz . . 14
3.4 Beispielsatz: Eine Konjunktion kommt mitten im Satz vor 16
3.5 Beispielsatz: Das Verb muss hinter zwei Adverbien platziert werden 16
3.6 Beispielsatz: Das Verb muss zwischen zwei Adverbien platziert werden . . . 17
3.7 Beispielsatz: Das Verb muss vor einem Adverb platziert werden 17
3.8 Falsche erkannte Zugehörigkeit des Artikels und Subjekts 20
3.9 Falsch erkannte Zugehörigkeit von „von“ bei falscher Benutzereingabe 20

Tabellenverzeichnis

4.1 Testset: Relative und absolute Anzahl an korrekten beziehungsweise falschen Benutzereingaben 25
4.2 Testset: Relativer und absoluter Anteil an korrekten beziehungsweise falschen Transformationen 26
4.3 Testset: Absolute Anzahl zwischen den Relationen von korrekten/falschen Sätzen zu Transformationen 31
4.4 Testset: Relativer und absoluter Anteil der vier Kategorien aus der Arbeit von Borodkin [Bor20] 32
4.5 Testset: An unserer Metrik angepasster relativer und absoluter Anteil der guten/schlechten Übersetzungen aus der Arbeit von Borodkin [Bor20] 33
4.6 Testset: Vergleich der als gut oder schlecht bewerteten Sätze aus der Arbeit von Borodkin nach unterschiedlichen Metriken 34
4.7 Testset: Absolute Anzahl der korrekt/falsch umgeformten Hauptsätze mithilfe von ParZu (Transformation) und der Methode von Borodkin (Übersetzung) nach unserer Metrik 35
4.8 Testset: Absolute Anzahl der von unserem Filter als richtig/falsch bewerteten Eingaben in Relation zu den Transformationen 38

Kapitel 1

Einleitung

Es gibt viele unterschiedliche Formen von Online-Partizipationen in denen verschiedene Benutzer miteinander interagieren können. Ein Teilgebiet dieser Partizipationen ist die Diskussion zwischen den Teilnehmern. Bestehende Lösungen lassen sich unter anderem in Foren und Listen gliedern. Allerdings hat sich gezeigt, dass es bei diesen Ansätzen zu Problemen kommt. Foren und Pro/Kontra-Listen sind für eine große Anzahl von Teilnehmern ungeeignet, da die Anzahl an Beiträgen proportional zu der Anzahl an Teilnehmern steigt. Zudem werden die Beiträge nicht miteinander in Bezug gebracht und die Darstellung ist dementsprechend unübersichtlich.

Eine neuere Variante, welche dieses Problem lösen soll und zeitgleich zugänglich für unerfahrene Benutzer sein soll, ist das Dialog-basierte Argumentationssystem D-BAS [Kra+18]. Bei diesem Ansatz wird versucht, einen zeitversetzten Dialog zu realisieren. Benutzer können Ihre Haltung zu einem Thema und Argumente für Ihre Position angeben. Außerdem werden dem Benutzer die Meinungen und Argumente anderer Teilnehmer zum gleichen Thema präsentiert. Diesen kann er zum Beispiel zustimmen, widersprechen oder korrigieren. Aufgrund der Entscheidung die der Benutzer getroffen hat, werden ihm im zukünftigen Verlauf der Diskussion geeignete Argumente und Aussagen angezeigt.

Beispielsweise können wir einen Ausschnitt eines solchen Diskussionsverlauf von D-BAS in Abbildung 1.1 betrachten. In der Diskussion geht es um folgende Fragestellung: „Wie können der Informatik-Studiengang verbessert und die Probleme, die durch die große Anzahl der Studierenden entstanden sind, gelöst werden?“ Aus einigen Vorschlägen zu diesen Thema, haben wir uns als Benutzer dafür entschieden, über das untergeordnete Thema „mathema-

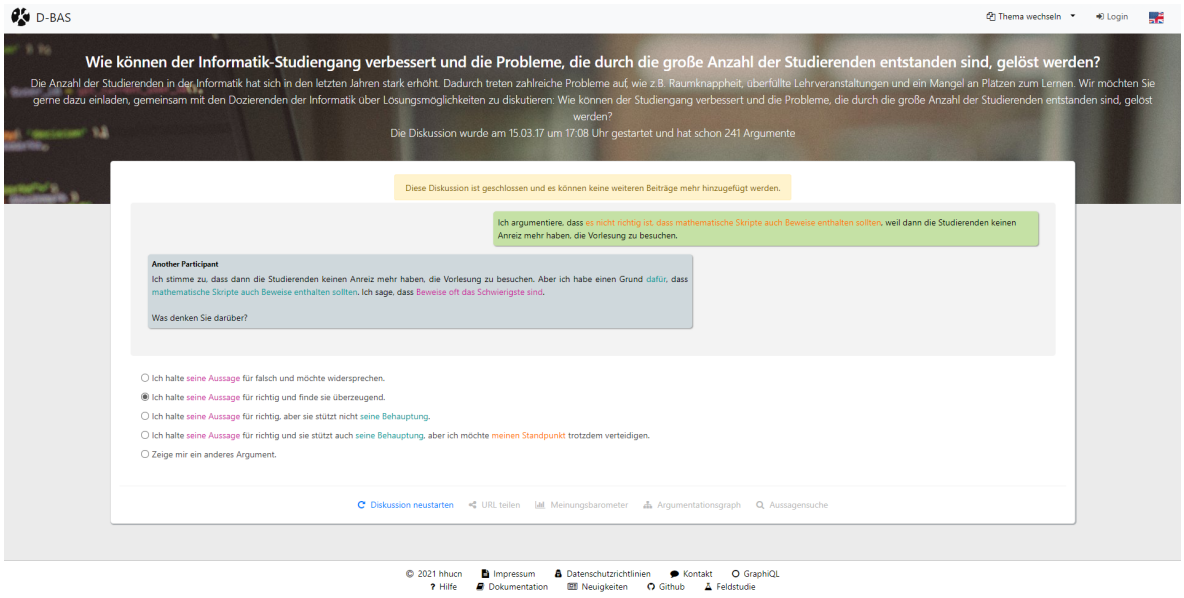

Abbildung 1.1: Screenshot von einer geschlossenen Diskussion aus D-BAS [Kra+18]

tische Skripte sollten auch Beweise enthalten“ zu reden. Wir finden es falsch, dass „mathematische Skripte auch Beweise enthalten sollten“ und stützen uns dabei auf eine Aussage von einem anderen Teilnehmer, welcher als Grund angegeben hat, dass sonst Studierende keinen Anreiz mehr haben, die Vorlesung zu besuchen. Wir werden nun mit einer Aussage von einem anderen Teilnehmer konfrontiert, welcher unserem Grund zustimmt, aber zusätzlich einen Grund für das Enthalten von Beweisen in mathematischen Skripten liefert. Dieser Teilnehmer behauptet, dass „Beweise oft das Schwierigste sind“ und deswegen diese in das Skript mitaufgenommen werden sollen. Wir haben nun verschiedene Möglichkeiten zu reagieren. Wir können die Aussage ablehnen, der Aussage zustimmen und unsere Meinung zum Thema ändern, der Aussage zustimmen, aber diese nicht relevant für die Behauptung („mathematische Skripte sollten auch Beweise enthalten“) halten, der Aussage zustimmen, diese für relevant halten, aber dennoch ein Gegenargument liefern oder wir können uns eine andere Aussage anzeigen lassen und darauf dann entsprechend reagieren.

D-BAS nimmt die Argumente der Benutzer in Form eines Nebensatzes an. Das Argument, welches ein anderer Benutzer angegeben hat, wird für den aktuellen Teilnehmer grammatikalisch nicht umgestellt. Das heißt, dass die Argumente von anderen Nutzern als Nebensatz präsentiert werden. Diese Darstellung ist bei der praktischen Anwendung eher hinderlich und unübersichtlich. Das Ziel dieser Arbeit ist es, einen Algorithmus zur automatischen Umformulierung von deutschen Neben- in Hauptsätzen zu implementieren und zu evaluieren.

Dadurch verschaffen wir den Benutzern einen besseren Überblick und die Benutzererfahrung der Teilnehmer wird verbessert. In dem obigen Beispiel (Abbildung 1.1) können wir uns die letzte Antwort von einem anderen Teilnehmer genauer anschauen. Momentan steht dort folgendes: „Aber ich habe einen Grund dafür, dass mathematische Skripte auch Beweise enthalten sollten. Ich sage, dass Beweise oft das Schwierigste sind.“ Wir können nun den Nebensatz „dass mathematische Skripte auch Beweise enthalten sollten.“ als Eingabe entgegen nehmen. Nach unserer Umwandlung in einem Hauptsatz erhalten wir: „Mathematische Skripte sollten auch Beweise enthalten.“ Die so umgeformte Aussage können wir nun mit der Behauptung verbinden und erhalten somit folgende Formulierung „Mathematische Skripte sollten auch Beweise enthalten, weil Beweise oft das Schwierigste sind.“, welche, unserer Meinung nach, viel angenehmer zu lesen und leichter zu verstehen ist, als die ursprüngliche Formulierung. Unabhängig davon können wir zumindest festhalten, dass der Abschnitt aus dem Text nach der Umformulierung weniger verschachtelt und einfacher aufgebaut ist.

1.1 Verwandte Arbeiten

Die Arbeit von Borodkin [Bor20] beschäftigt sich mit der Umwandlung von Neben- in Hauptsätzen. In der Arbeit werden deutschsprachige Sätze in englischsprachige Sätze übersetzt und anschließend wieder in die deutsche Sprache umgewandelt. Die Idee hinter diesem Vorgehen ist es, die englische Satzstellung auszunutzen. Im Englischen unterscheidet sich die Satzstellung zwischen Neben- und Hauptsätzen nicht. Somit soll das Verb beim Übersetzten automatisch in die richtige Position im Satz gesetzt werden und anschließend bei der erneuten Umwandlung in die Deutsche Sprache an der richtigen Stelle bleiben. Andere Arbeiten, welche sich mit der Umwandlung von deutschsprachigen Neben- in Hauptsätzen auseinandersetzen, wurden bis zum jetzigen Zeitpunkt nicht veröffentlicht und sind uns nicht bekannt.

In dieser Arbeit wird ein anderer Ansatz untersucht. Wir erforschen inwieweit es möglich ist, die Umwandlung von Neben- in Hauptsätzen unter anderem mit Hilfe von POS-Taggern und Dependency-Parsern umzusetzen. Es existieren sowohl eine Vielzahl von POS-Taggern, wie zum Beispiel clevertagger [SVS13], JCoRe Base [UHH16], Apache OpenNLP [Ope11], NLTK [SL08], SoMeWaTa [Pro18] und TreeTagger [Sch13], als auch Dependency-Parser, wie zum Beispiel ParZu [Sen+09], CoreNLP [Man+14], einen Parser für medizinische Texte [Kar+18], spaCy [Hon+16], MSTParser [McD+05], MaltParser [NHN06] und Stanford-

NLP [Zem+18]. Wir werden die Bedeutung der beiden Begriffe in Kapitel 2 näher erläutern.

Ein weiteres Anwendungsgebiet von POS-Taggern und Dependency Parsern ist die automatische Umformulierung und Vereinfachung von komplexen Sätzen in der deutschen Sprache. In diesem Anwendungsgebiet versucht man, Wortgruppen und einzelne Abschnitte zu identifizieren und den Text entsprechend in eine einfachere und leichtere Form umzuwandeln. Dabei werden Nebensätze nicht gezielt in Hauptsätze umgewandelt, sondern die Eingabesätze werden generell umgeschrieben, wobei teilweise auch andere Formulierungen eingebaut werden. Es existieren einige solcher Online-Textumschreibung-Tools, wie zum Beispiel [Smo], [Men] oder [Pre].

1.2 Aufbau der Arbeit

In Kapitel 1 haben wir festgestellt, inwiefern eine automatische Umwandlung von Neben- in Hauptsätzen zur Verbesserung der Benutzererfahrung in Online-Diskussionsforen beiträgt und ein generelles Verständnis für die Problemstellung erhalten. Wir können diese Arbeit in bestehende Literatur einordnen und wissen, in welchen Aspekten sich diese Arbeit von den bestehenden Werken abgrenzt.

Als Erstes gehen wir auf die Grundlagen ein, die wir für die Implementierung und Auswertung des Algorithmus brauchen. Dort werden wir die benötigten Begriffe und die allgemeine Vorgehensweise kurz erläutern. Im nächsten Kapitel 3 wird die Implementierung vorgestellt und erklärt. Dabei gehen wir auch auf die Schwierigkeiten während der Implementierungsphase ein und stellen einige solcher speziellen Fälle individuell vor. Im anschließenden Kapitel 4 werden die Ergebnisse evaluiert und unter anderem mit anderen bekannten Ergebnissen verglichen. Basierend darauf wird im letzten Kapitel 5 ein Fazit gezogen und ein Ausblick auf weitere Entwicklungsmöglichkeiten und generelle zukünftige Arbeiten gegeben.

Kapitel 2

Grundlagen

In Kapitel 2 erläutern wir die benötigten Grundlagen für die Implementierung und für die Evaluierung genauer. Dabei gehen wir auf allgemeine Methodiken in diesem Anwendungsgebiet ein und führen bewährte Ansätze zur automatischen Erkennung von Wortarten und deren Funktion in Sätzen auf.

2.1 Testsets und Trainingssets

Ein wichtiges Element bei der Entwicklung eines Algorithmus für die automatische Umwandlung von Neben- in Hauptsätze und generelles Testen von Algorithmen sind die Test- und Trainingssets. Trainingssets sind Datenmengen, welche uns bei der Entwicklung und Implementierung des Algorithmus zur Verfügung stehen. Die Hauptaufgabe dabei ist es, so viele Daten wie möglich mithilfe des Algorithmus korrekt zu verarbeiten. In unserem spezifischen Fall sind die beiden Datensätze Mengen von Nebensätzen aus zwei Feldexperimenten von D-BAS [KMM17], welche wir mit unserem Algorithmus in Hauptsätze umwandeln.

Während des Entwicklungsprozesses können wir die Resultate von dem Trainingsset verwenden, um unseren Algorithmus im Bezug auf die Ergebnisse aus dem Trainingsset zu verbessern und somit ein besseres Ergebnis in den darauffolgenden Trainingsdurchläufen zu erzielen. Vereinfacht ausgedrückt bedeutet dies, dass wir abwechselnd die gleiche Datenmenge auswerten und anschließend den Algorithmus basierend auf den Ergebnissen verbessern. Dieser Schritt wird solange wiederholt, bis im besten Fall die gesamte Datenmenge

des Trainingssets korrekt verarbeitet wird oder bis keine weiteren vernünftigen Verbesserungen vorgenommen werden können. Unter vernünftigen Verbesserungen verstehen wir solche Verbesserungen, die voraussichtlich nicht nur zu einem verbesserten Ergebnis der Auswertung des Trainingsset führen, sondern auch zu einer verbesserten Verarbeitung des Testsets und allgemeinen Datensätzen führen. Zum Beispiel zählt eine händische Regel, welche überprüft, ob ein konkreter Satz vorliegt und dann eine vordefinierte Ausgabe generiert, nicht zu den vernünftigen Verbesserungen. Viel mehr versuchen wir Regeln zu erzeugen, welche allgemeingültig sind.

Das Testset hingegen ist für die Evaluierung des Algorithmus zuständig. Diese Datenmenge ist uns während der Entwicklung des Algorithmus unbekannt und wird auch währenddessen nicht benutzt. Somit können wir ausschließen, dass der Algorithmus auf dieses Testset abgeschnitten ist, nur dort gute Resultate erzielt und bei anderen Datenmengen erheblich schlechter funktioniert. Sobald der Algorithmus evaluiert werden soll, werden die Ergebnisse für das Testset kalkuliert. Anhand der Resultate können wir die Güte des Algorithmus bestimmen. Wichtig hierbei ist es, dass wir die Erkenntnisse aus den Resultaten nicht für die weitere Entwicklung des Algorithmus benutzen dürfen, solange wir das Testset beibehalten. Wenn der Algorithmus allerdings nach der Auswertung weiterentwickelt werden soll, kann das Testset zum Trainingsset umgewandelt werden. Für die neue Evaluierung brauchen wir dann aber ein neues und wiederum unbekanntes Testset.

2.2 Part-Of-Speech Tagging

Part-Of-Speech-Tagging (POS-Tagging) weist jedem Wort und Satzeichen eines Satzes eine Wortart zu. Dabei wird sowohl das Wort an sich betrachtet, als auch der Kontext des Wortes im Satz. Dies erfolgt heutzutage meist automatisch und im Laufe der Zeit wurden einige unterschiedliche POS-Tagger entwickelt. Diese unterscheiden sich hauptsächlich in natürlicher Sprache, Format, Anwendungsgebiet und Interpretation des Kontextes. Tagger benutzen hauptsächlich feste grammatikalische und zum Teil zusätzlich handgeschriebene Regeln. Man unterscheidet zudem zwischen POS-Taggern, welche mit und ohne einem getaggten Trainingskorpus arbeiten. Tagger welche einen Korpus verwenden, können aufgrund der vorliegenden Ergebnisse zusätzlich zum Beispiel die Wahrscheinlichkeit für aufeinanderfolgende Wortarten bestimmen und dieses Wissen in das Tagging mit einbringen. Es existieren auch Projekte wie zum Beispiel CoreNLP [Man+14], spaCy [Hon+16] und Stanford-

NLP [Zem+18], welche sowohl POS-Tagger als auch Dependeny Parser implementieren.

2.3 Dependency Parser

Dependency Parser untersuchen die Struktur und Grammatik von Wörtern und Abschnitten in einem Satz. Das Ziel ist es, die Abhängigkeiten von Wörtern und Abschnitten von Sätzen innerhalb eines Satzes auf den Kontext angewandt anzugeben. Man versucht, zusammengehörige Wörter, Worte welche voneinander abhängig sind und ganze Neben- und Hauptsätze zu erkennen. Zudem soll die Semantik und der Syntax eines Satzes erkannt und wiedergegeben werden. Diese Aufgabe ist sehr komplex und nach ausführlicher Recherche ist bis zum jetzigen Zeitpunkt keine Implementierung bekannt, welche diese Problemstellung mit einer Genauigkeit von 100 % lösen kann. Die verschiedenen Dependency Parser sind meistens auf bestimmte Themengebiete zugeschnitten und erreichen dementsprechend für unterschiedliche Eingabemengen unterschiedliche Erfolgsquoten. Zum Beispiel spezialisiert sich das Werk [Kar+18] auf medizinische Texte und ist in diesem Umfeld besonders gut geeignet. Wir werden noch im darauffolgenden Kapitel 3 untersuchen, inwieweit diese Ungenauigkeit Einfluss auf die Transformation von Neben- in Hauptsätzen hat.

Eine übersichtliche Darstellung für die Abhängigkeiten innerhalb eines Satzes bietet der Abhängigkeitsgraph. Jedes Wort repräsentiert einen Knoten und die Abhängigkeiten zwischen den Worten beziehungsweise Knoten werden mit gerichteten Kanten dargestellt. Satzzeichen werden nicht als Knoten an sich bewertet, werden aber dennoch für die Erkennung von Abschnitten innerhalb eines Satzes herangezogen. In der Abbildung 2.1 ist ein solcher Beispielgraph zu sehen. Die „oberste Schicht“ (Wurzel) einer solchen Abhängigkeitsdarstellung ist das Verb des Hauptsatzes, welches in diesem Fall das „ist“ ist. Von dort aus werden die weiteren Abhängigkeiten durch Kanten dargestellt. Beispielsweise wird die Kante zwischen dem letzten Teil des Hauptsatzes und dem Verb des Einschubs durch ein „rel“ gelabelt. Dies steht für einen Relativsatz und dort bildet, wie beim Hauptsatz auch, ebenfalls das Verb die oberste Schicht.

Um einen solchen Graphen zu erstellen und generell Dependency Parsing durchzuführen, wird unter anderem auf POS-Tagging zurückgegriffen. Auch in diesem Gebiet gibt es einige automatisierte Algorithmen, die wir schon in Kapitel 1 aufgeführt haben.

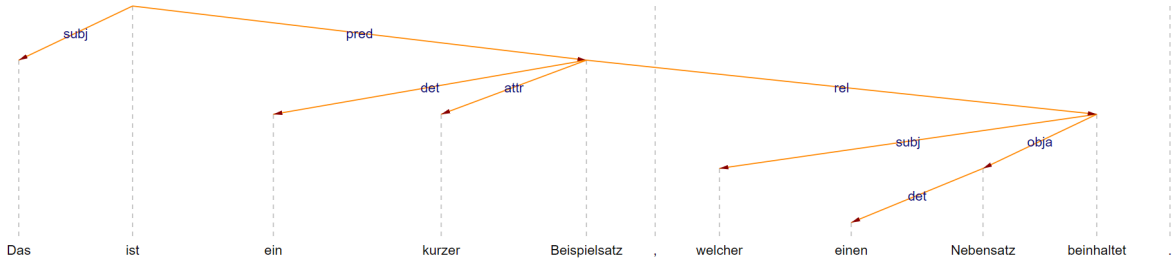

Abbildung 2.1: Grafik mit ParZu generiert [Sen+09]: Beispielsatz mit zugehörigem Graphen

Wir kennen nun die Grundlagen für die Entwicklung eines Algorithmus zur automatischen Umformulierung von Neben- in Hauptsätzen. Als Nächstes werden wir diese Grundlagen nutzen, um geeignete Tools auszusuchen und einen Algorithmus für eine solche Umwandlung zu implementieren. Des Weiteren werden wir den Einfluss der Ungenauigkeit von POS-Taggern und Dependency Parsern intensiver behandeln.

Kapitel 3

Implementierung

Bevor wir mit der Implementierung beginnen können, müssen wir die zu verwendenden Tools und Datensätze (Trainingsset) festlegen. Da diese Arbeit für D-BAS entwickelt wird, werden für das Trainingsset bereits bestehende Nutzereingaben der zwei Feldexperimente aus dem Online-Diskussionsforum [KMM17] verwendet. Zeitgleich wird eine bessere Vergleichsbasis zu der bereits bestehenden Arbeit [Bor20] geschaffen, indem dasselbe Trainings- und Testset verwendet wird. Nicht zuletzt bietet dieser Datensatz dank seines Ursprungs eine realistische Umgebung.

Die Auswahl der geeigneten Tools gestaltet sich etwas schwieriger. Wir können uns folgenden Nebensatz genauer anschauen: „Ich bin dieser Meinung, weil das Tragen von Masken sinnvoll ist.“ Unser Ziel ist es, den zweiten Teil des Satzes, also den Nebensatz, in einen Hauptsatz umzuwandeln. In der deutschen Sprache unterschiedet sich die Satzstellung zwischen einem Neben- und Hauptsatz in nahezu fast allen Fällen nur in der Positionierung des Verbs. Demnach ist eine der Hauptaufgaben, das Verb bei einer Umformulierung von einen Neben- in einem Hauptsatz umzustellen. Außerdem müssen wir die weitere Satzstellung untersuchen, damit wir das Verb an der richtigen Stelle positionieren können. Beispielsweise muss das Verb nach dem Subjekt und dem dazugehörigen Zusatz platziert werden, aber muss vor dem Akkusativobjekt vorkommen.

Generell können wir feststellen, dass ein POS-Tagger diese Aufgabe alleine nicht lösen kann. Wir müssen erkennen können, welcher Teil im Satz zum Subjekt gehört und welcher Abschnitt im neuen Satz einen Hauptsatz und welcher einen Nebensatz bildet. Im ersten Schritt versuchen wir die Satzstruktur zu erkennen, um danach im zweiten Schritt grammatikalische

und handgeschriebene Regeln auf die gesammelten Informationen anzuwenden.

In unserem Beispiel soll der Algorithmus feststellen, dass „das Tragen“ als Subjekt fungiert, „von Masken“ zum Subjekt gehört, „sinnvoll“ ein auf das Verb bezogenes Adjektiv ist und demnach im Hauptsatz hinter dem Verb positioniert werden muss und letztlich das Verb „ist“ erkennen, welches als „Hauptverb“ im neuen Satz agiert und umgestellt werden soll. Unser Algorithmus muss nun anhand von Regeln entscheiden, wo im neu transformierten Satz das Verb platziert werden soll und folgendes Ergebnis ausgeben: „Das Tragen von Masken ist sinnvoll.“ In Kapitel 2 haben wir den Dependency Parser als ein Werkzeug kennen gelernt, welches Satzstrukturen erkennen kann. Nun stellt sich die Frage, welcher Dependency Parser am geeignetsten ist. Wir haben bereits in Abschnitt 1.1 dieser Arbeit erläutert, dass noch keine Veröffentlichung zu diesem spezifischen Thema vorliegt. Allerdings existieren Arbeiten, welche die Effizienz und die Genauigkeit verschiedener Parser vergleichen [ORD19]. In der dortigen Auswertung erzielt ParZu [Sen+09] die besten Ergebnisse bezüglich der Zugehörigkeiten („Dependency“) der Wörter innerhalb eines Satzes. Zusammen mit dem von den Entwicklern von ParZu vorgeschlagenen POS-Tagger clevertagger [SVS13] wird dieser Dependency Parser für die Entwicklungsphase beziehungsweise Implementierung verwendet.

3.1 CoNLL-Format

Das Ausgabeformat von ParZu ist das CoNLL-Format [Niv+07]. Dieses Format weist jedem Wort im Satz zehn Felder zu, wobei für uns sieben relevant sind. Das erste Feld „ID“ ordnet jedem Wort einen Index (beginnend mit eins) aufsteigend zu. Das zweite Feld „FORM“ beinhaltet das Wort an sich und das vierte Feld „UPOS“ gibt den universellen POS-Tag wieder. Feld fünf „XPOS“ hingegen beinhaltet den sprachspezifischen POS-Tag. Die morphologischen Informationen sind in Feld sechs „FEATS“ gespeichert oder ist gleich „_“, falls keine morphologischen Informationen vorhanden sind. In Feld sieben „HEAD“ befindet sich der Kopf/Zeiger des aktuellen Wortes. Der Wert ist gleich der ID des korrespondierenden Wortes oder null, wenn keine Abhängigkeit erkannt wird. Das letzte relevante Feld acht „DEPREL“ gibt die universelle Abhängigkeitsrelation des Wortes zum Kopf an oder ist gleich „root“, falls der Kopf gleich null ist. Die Informationen, die in diesen Feldern gespeichert werden, sind für die Umformung des Satzes essentiell. Diese Daten bilden den Grundstein der Implementierung. Als Nächstes erläutern wir, wie wir die Daten aus dem Dependency Parser in

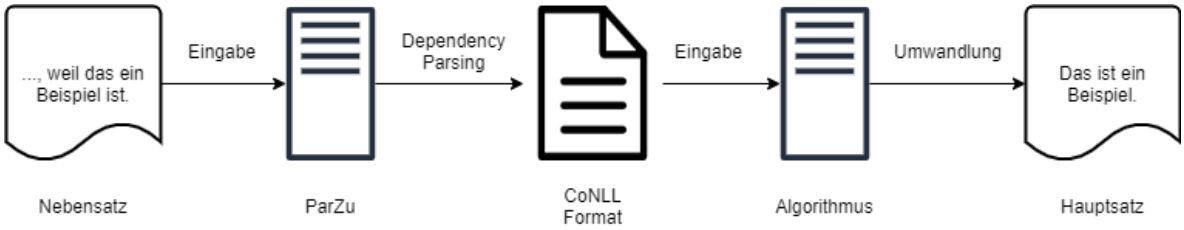

Abbildung 3.1: Überblick über den gesamten Ablauf der Transformation von Neben- in Hauptsätzen

der Implementierung benutzen.

3.2 Konvertierung des Nebensatzes in einen Hauptsatz

In Abschnitt 3.2 beschreiben wir das generelle Vorgehen bei der Umwandlung von einem Nebensatz in einen Hauptsatz. Wir nehmen an dieser Stelle an, dass die Eingabe korrekt ist. Das heißt, dass Grammatik, Rechtschreibung, Semantik und Syntax richtig sind. Ein Beispiel für eine korrekte Eingabe ist der Satz „Ich bin dieser Meinung, weil in der Informatik das freie Sprechen nicht ausführlich genug geübt wird.“ Im Gegensatz dazu ist eine mögliche Antwort auf diese Aussage „Ich bin dieser Meinung, weil dies ist nciht die Aufgabe der Universität.“ grammatikalisch nicht korrekt. Das Verb ist fälschlicherweise wie in einem Hauptsatz positioniert, obwohl das Argument in Form eines Nebensatzes angegeben wird. Zudem ist im Wort „nciht“ ein Rechtschreibfehler vorhanden.

Außerdem gehen wir davon aus, dass die Eingabedatei einzelne Sätze zeilenweise trennt. Ferner nehmen wir an, dass ParZu jeden Satz korrekt verarbeitet, also alle Felder richtig zugewiesen sind. Der allgemeine Ablauf sieht wie folgt aus (siehe Abbildung 3.1):

Anfangs erhalten wir einen Datensatz aus D-BAS, welcher Benutzereingaben in Form von Nebensätzen beinhaltet. Wir gehen den Datensatz zeilenweise durch und entfernen alle Punkte, welche nicht am Ende der Zeile stehen. Dies ist notwendig, da ParZu einen Punkt mit einem anschließenden Leerzeichen als Satzende interpretiert und so Abkürzungen mit einem Punkt nicht verarbeiten kann und fehlinterpretiert. Um einen kompletten Satz zu bilden, fügen wir als Präfix des Satzes „Ich bin dieser Meinung, weil“ hinzu. Dies wird benötigt, da ParZu einen vollständigen Satz als Eingabe braucht. Danach wandelt ParZu den Datensatz um und wir erhalten als Ausgabe das oben beschriebene CoNLL-Format. Dieses For-

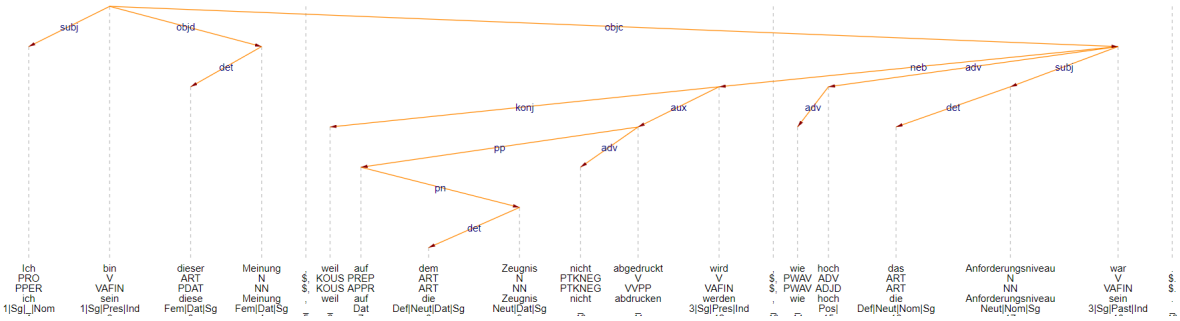

Abbildung 3.2: Grafik mit ParZu generiert [Sen+09]: Ursprünglicher Nebensatz (Trainingsset) wird während der Transformation in Haupt- und Nebensatz aufgeteilt

mat enthält die benötigten Informationen, um nun unseren Algorithmus auf die Datenmenge anzuwenden. Als Ausgabe erhalten wir die umgewandelten Hauptsätze. Wie genau unser Algorithmus funktioniert wird im folgenden beschrieben (siehe Abbildung 3.3):

Wir nehmen die Eingabe entgegen und gehen den Datensatz für jeden einzelnen Satz durch. Als erstes speichern wir alle Felder des CoNLL-Formats ab. Die folgenden Schritte basieren alle auf den Informationen aus den Feldern des CoNLL-Formats. Zuerst überprüfen wir, ob eine korrekte Benutzereingabe vorliegt. Dies heißt nicht, dass wir prüfen, ob die Eingabe grammatikalisch korrekt ist, sondern ob die vorherigen Schritte korrekt durchgeführt worden sind. Daraufhin geht der Algorithmus den Satz Wort für Wort beziehungsweise Feld für Feld durch. Sobald wir eine Unterbrechung des aktuellen Satzteils finden, wird untersucht, ob der Satzteil bis dorthin im neuen transformierten Satz Teil des Hauptsatzes oder Teil des Nebensatzes ist. Mit einer Unterbrechung meinen wir zum Beispiel einen Einschub, einen Nebensatz oder eine Aufteilung des aktuellen Satzes durch ein „und“ oder „oder“.

An dieser Stelle sei kurz anzumerken, dass nicht jedes solcher Vorkommen als Unterbrechung gewertet wird. Beispielsweise kann ein „und“ in einer Aufzählung vorkommen, sodass wir in diesem Fall den Satz dort nicht trennen wollen. Falls auch im umgeformten Satz der aktuelle Satzteil einen Nebensatz bildet, übernehmen wir den originalen Teil und fahren fort. Falls der Satzteil im transformierten Satz zu einem Hauptsatz wird, konvertieren wir diesen Abschnitt in einen Hauptsatz, indem wir das richtige Verb an die korrekte Position setzen. Somit bauen wir schrittweise den neuen Hauptsatz auf. Wenn wir am Ende des Satzes angekommen sind, erhalten wir zum korrespondierenden Nebensatz den zugehörigen Hauptsatz.

Nehmen wir als Beispiel den Satz „Ich bin dieser Meinung, weil auf dem Zeugnis nicht abgedruckt wird, wie hoch das Anforderungsniveau war.“ (Abbildung 3.2) aus dem Trainingsset.

Offensichtlich ist sowohl der Teil „auf dem Zeugnis nicht abgedruckt wird“ als auch „wie hoch das Anforderungsniveau war“ Teil des Nebensatzes. Der Algorithmus erkennt nun anhand des Kommas und weiteren damit verbundenen Regeln, dass dort der erste Teil unterbrochen wird. Weiterhin erkennt der Algorithmus, dass dieser Teil im zu erstellenden Satz zum Hauptsatz gehört und dementsprechend wird der Abschnitt zu „auf dem Zeugnis wird nicht abgedruckt“ transformiert. Der zweite Teil hingegen bleibt auch im neuen Satz ein Nebensatz und wird folgerichtig angehangen. Wir erhalten somit folgende Umwandlung: „Auf dem Zeugnis wird nicht abgedruckt , wie hoch das Anforderungsniveau war.“ Um die Ergebnisse aus dem Trainingsset und Testset besser auswerten zu können, ist sowohl vor Satzzeichen als auch nach Satzteichen jeweils ein Leerzeichen eingefügt worden. Die Formatierung wird in dieser Arbeit übernommen und hat keinen Einfluss auf die Güte des Algorithmus.

In der Praxis sind die Benutzereingaben nur selten ohne kleinere oder größere Fehler. Auch während des Taggings, des Parsens oder der Umwandlung können Fehler auftreten. Manche sind unabhängig voneinander, andere hingegen sind die Folge von vorangegangenen Fehlern. Es gibt dabei oft viele unterschiedliche Fehler und einige Fehlerquellen während des gesamten Prozesses. Daraus folgt, dass der oben beschriebene Prozess in dieser Form in der Praxis nicht funktioniert. Im richtigen Einsatz müssen wir versuchen, auch fehlerhafte Sätze umzuwandeln, wobei das Erkennen solcher nicht immer einfach ist. Um die „fehlerhaften“ Sätze umwandeln zu können, müssen einige besondere Regeln implementiert und auf viele Besonderheiten geachtet werden. Wir können schon an dieser Stelle kurz festhalten, dass wir nicht alle Fehler beheben können. Betrachten wir den Beispielsatz vom Anfang des Abschnittes 3.2: „Ich bin dieser Meinung, weil dies ist nciht die Aufgabe der Universität.“ Wir können den Rechtschreibfehler nicht beheben, aber unser Algorithmus kann erkennen, dass das Verb schon für einen Hauptsatz an der richtigen Stelle steht und wird dementsprechend folgenden Hauptsatz ausgeben: „Dies ist nciht die Aufgabe der Universität.“ Einige Regeln zur Transformation von Neben- in Hauptsätze werden wir in Abschnitt 3.3 genauer erklären.

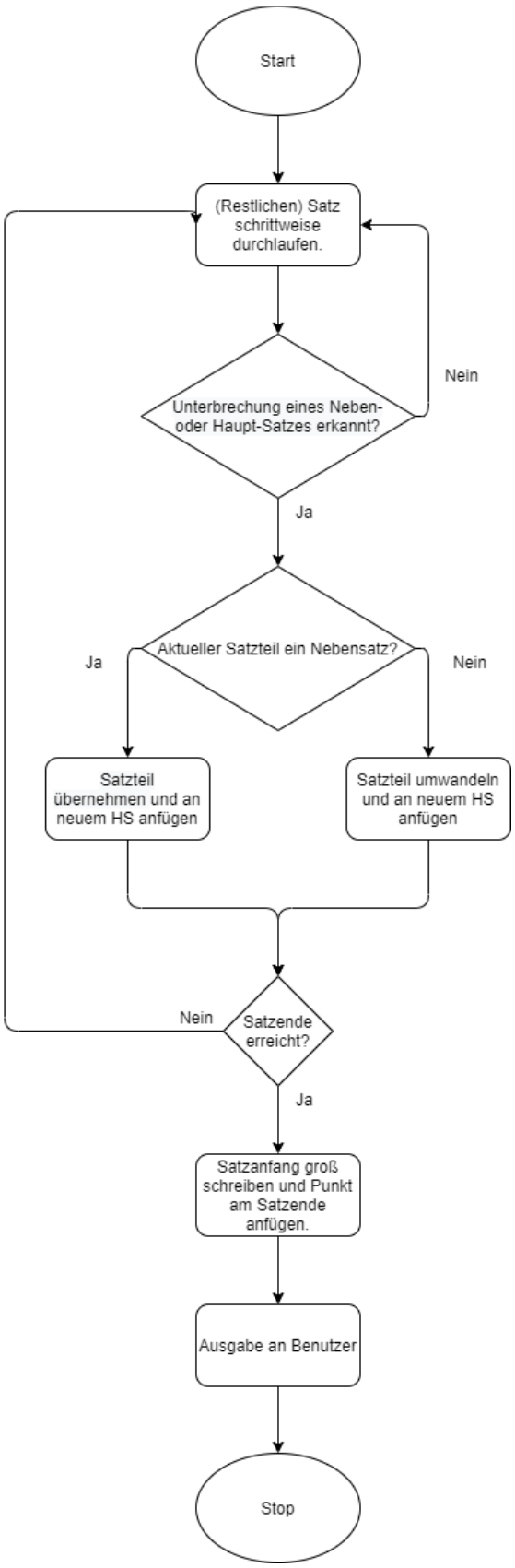

Abbildung 3.3: Programmablauf der Transformation vom CoNLL-Format zum Hauptsatz

3.3 Abweichungen vom Idealfall und zusätzliche Sonderfälle

Während der Implementierung und Auswertung des Trainingssets sind einige Abweichungen von einem perfekten Satz (ohne Fehler) aufgefallen. In Abschnitt 3.3 gehen wir auf einige dieser besonderen Fälle ein.

3.3.1 Allgemeine Problemstellung

Bevor wir auf die spezifischen Fälle genauer eingehen, erläutern wir an dieser Stelle den Einfluss der Fehler auf die Implementierung. Im perfekten Fall sind alle Felder im CoNLL-Format aller Wörter im Satz korrekt ausgefüllt. Eine genauere Ausführung der Bedeutungen der Felder im Fall von XPOS und DEPREL sind in der Arbeit [Fot06] zu finden. Falls der Satz nun korrekt geparst wurde, können wir anhand der Informationen feste Regeln für die Umformulierung festsetzen. Das perfekte Parsing ist allerdings die Ausnahme und somit werden die Felder teils falsch und teils gar nicht ausgefüllt. Die Regeln, die vorher funktioniert haben, würden den Satz jetzt nicht richtig transformieren. Wir können daraufhin die Regeln anpassen, allerdings kann schon im nächsten Schritt der nächste Fehler zu einer Belegung der Felder führen, welche diese Regel auch „falsch“ macht. Der Kompromiss zwischen dem richtigen Transformieren der aktuellen Sätze, der Beibehaltung der korrekten Umformulierung von bereits behandelten Sätzen und der Rücksichtnahme auf zukünftige Sätze ist eine der größten Aufgaben während der Implementierung. Durch diese massive Komplexität würde eine genaue Erklärung der im folgenden erwähnten Regeln schnell unübersichtlich werden. Deshalb versuchen wir, eine grobe Zusammenfassung der Ideen, Funktionen und Schwierigkeiten der spezifischen Regeln aufzuzeigen.

Wir sprechen von Hauptsätzen, wenn im transformierten Teil der ehemalige Nebensatz zum neuen Hauptsatz wird, und wir bezeichnen einen Abschnitt als Nebensatz, wenn der aktuelle Teil des Satzes auch im transformierten Satz ein Nebensatz bleibt.

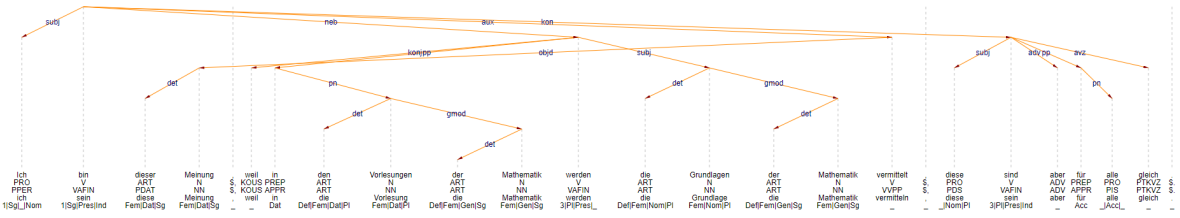

Abbildung 3.4: Grafik mit ParZu generiert [Sen+09]: Beispielsatz: Eine Konjunktion kommt mitten im Satz vor

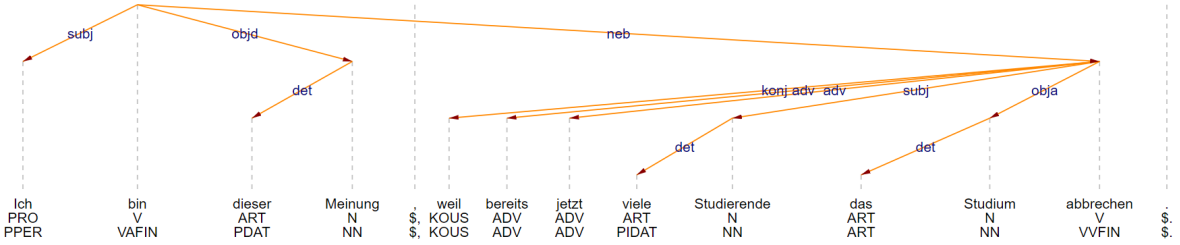

Abbildung 3.5: Grafik mit ParZu generiert [Sen+09]: Beispielsatz: Das Verb muss hinter zwei Adverbien platziert werden

3.3.2 Konkrete Sonderfälle

Wie in Abschnitt 3.2 beschrieben, wird der Satz in Abschnitte unterteilt. Um diese Abschnitte erkennen zu können und zu entscheiden, ob dieser Teil zum Hauptsatz wird, oder auch im transformierten Teil ein Nebensatz bleibt, werden die Kommata (und andere Satzzeichen) im Satz genauer untersucht. Solche können sowohl einen Nebensatz als auch einen Hauptsatz einleiten beziehungsweise unterbrechen. Die meisten Nebensätze lassen sich erkennen, indem untersucht wird, ob nach einem Komma eine Konjunktion folgt. Bei der weiteren Untersuchung ist aufgefallen, dass bei einigen Sätzen Konjunktionen auch erst später vorkommen können. Oft passiert dies bei einem „aber“. Hier gibt es generell einige unterschiedliche Variationen, welches die Position des Wortes im Satz betrifft. Unabhängig davon, ob die Positionierung richtig ist oder nicht, existieren genug Fälle, sodass wir diesen Sonderfall separat behandeln. In Abbildung 3.4 ist ein Beispiel für einen solchen Fall aus dem Trainingsset aufgeführt. Dort steht das „aber“ nicht direkt hinter dem Komma, sondern mitten im Nebensatz.

Die Positionierung von Adverbien und Verben im neuen Hauptsatz ist ebenfalls eine Herausforderung. Abhängig von der Art des Adverbs, dem Vorkommen weiterer Adverbien und den anderen Wortarten im Satz wird das Verb entweder davor, dazwischen oder sogar da-

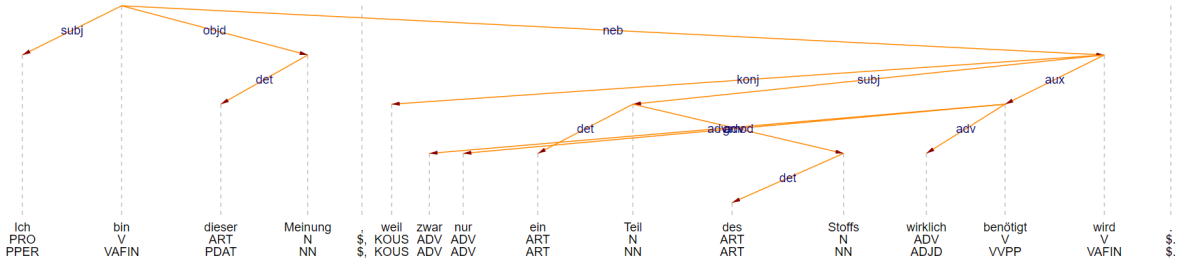

Abbildung 3.6: Grafik mit ParZu generiert [Sen+09]: Beispielsatz: Das Verb muss zwischen zwei Adverbien platziert werden

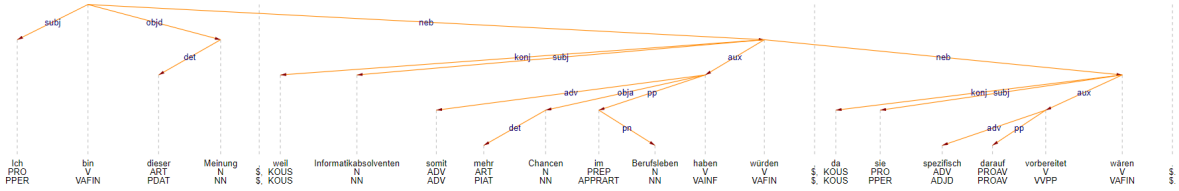

Abbildung 3.7: Grafik mit ParZu generiert [Sen+09]: Beispielsatz: Das Verb muss vor einem Adverb platziert werden

hinter platziert. Beispiele für alle drei Fälle finden wir im Trainingsset: Im ersten Satz „Ich bin dieser Meinung, weil bereits jetzt viele Studierende das Studium abbrechen.“ (Abbildung 3.5) muss das Verb in der Umformung hinter den beiden Adverbien platziert werden. Im zweiten Satz „Ich bin dieser Meinung, weil zwar nur ein Teil des Stoffs wirklich benötigt wird.“ (Abbildung 3.6) hingegen, müssen wir das Verb zwischen den beiden Adverbien „zwar“ und „nur“ setzen. Im dritten Satz „Ich bin dieser Meinung, weil Informatikabsolventen somit mehr Chancen im Berufsleben haben würden, da sie spezifisch darauf vorbereitet wären.“ (Abbildung 3.7) platzieren wir das Verb vor dem Adverb „somit“, da ein Subjekt vorangestellt wird.

Ein kleiner Sonderfall bildet noch das Vorkommen von Präpositionalphrasen nach einem Komma. Falls vor dem Abschnitt mit diesem Vorkommen ein Hauptsatz steht, wandeln wir diesen Abschnitt nicht um, da er schon im originalen Teil korrekt angeordnet ist. Betrachten wir zum Beispiel folgenden Satz: „Ich bin dieser Meinung, weil man so in den mathematischen Fächern auch die Möglichkeit hat, im Forum zu schreiben.“ Das „im“ nach dem Komma ist die Präpositionalphrase und bei der Umwandlung möchten wir den dazugehörigen Teil des Satzes nicht verändern. Wir unterteilen den Satz also an dieser Stelle, transformieren den ersten Abschnitt, fügen den zweiten an und erhalten folgenden Satz: „Man hat so in den mathematischen Fächern auch die Möglichkeit , im Forum zu schreiben.“

Um einiges schwerer ist das Erkennen von Hauptsätzen. In den meisten Fällen ist der erste Abschnitt im alten Satz der Hauptsatz in der transformierten Form. Doch es kann auch sein, dass zuerst ein Nebensatz eingeschoben wird und anschließend erst der Hauptsatz folgt. Es gibt keine typische Einleitung von Hauptsätzen und die Satzstellung ist im originalen Teil identisch zur Satzstellung der Nebensätze. Auch das Fehlen der Einleitung eines Nebensatzes ist kein ausreichendes Kriterium, wenn wir zum Beispiel Einschübe berücksichtigen. Diese beginnen meist nicht mit einer Konjunktion oder ähnlichem, bilden aber dennoch keine Hauptsätze. Wenn wir den genannten Kompromiss zwischen der Erstellung neuer Regeln und der Beibehaltung der korrekten Umwandlungen berücksichtigen, ist nur eine sehr strenge Bedingung für die Erkennung der Hauptsätze möglich, damit Sätze, welche nicht in diese Kategorie fallen, auch nicht nach der Regel behandelt werden. Wir haben für das Testen eigene Sätze erstellt, da im Trainingsset lediglich falsche Sätze mit einer solchen Satzstruktur vorliegen. In dem Datensatz fehlt zumindest immer die Kommasetzung, weswegen ein sinnvolles Testen nicht möglich ist. Beispielsweise fehlt im folgenden Satz ein Komma vor dem „wenn“ und der zweite Teil des Nebensatzes enthält eine falsche Satzstellung, da das Verb „sind“ nach hinten gestellt werden muss: „Ich bin dieser Meinung, weil wenn die Lösungen nur hochgeladen werden, sind die Studenten auch auf sich gestellt.“ Um eine solche Satzstruktur richtig verarbeiten zu können, haben wir eigene Beispielsätze erstellt, welche grammatikalisch korrekt sind.

Eine weitere Herausforderung ist die korrekte Positionierung des Verbs im Hauptsatz. In den meisten Fällen ist das Verb, welches wir umstellen möchten, das letzte Verb im Satz. Aber auch hier gibt es viele Ausnahmen. Bei der Erkennung des richtigen Nebensatzes versuchen wir zu erkennen, an welchen Stellen die Belegung der Felder des CoNLL-Formats richtig ist, an welchen Stellen sie teilweise richtig ist und an welchen Stellen lediglich die Erkennung des Verbs an sich korrekt ist. Darauf aufbauend versuchen wir Regeln zu erstellen, ohne die nicht zutreffenden Fälle mit einzuschließen und somit ein falsches Ergebnis zu verhindern. Unter anderem berücksichtigen wir an dieser Stelle das sprachspezifische Tagging, den DEPREL, den HEAD, den Numerus des Verbs und des Subjekts und den Vorgänger im Satz. Nach der Erkennung des Verbs muss noch festgelegt werden, ob das Verb ein Präfix besitzt und falls ja, ob es sich um ein trennbares oder nicht trennbares Präfix handelt. Da in der deutschen Sprache zum Beispiel „durch“ oder „unter“ sowohl trennbar als auch nicht trennbar sein können, stoßen wir an dieser Stelle an die Grenzen des Algorithmus. Um unterscheiden zu können, ob diese bestimmten Präfixe in der konkreten Situation vom Verb getrennt werden, müsste eine Kontextanalyse oder andere weiterführende Techniken implementiert

werden. Der Umfang der korrekten Einbindung dieser spezifischen Präfixe geht über den zeitlichen Rahmen dieser Arbeit hinaus.

Hauptsätze können mit Konjunktionen wie „und“ oder „oder“ unterteilt werden. In diesem Fall muss - falls vorhanden - in jedem Teil das Verb umgestellt werden. An dieser Stelle müssen wir unterscheiden, ob jeder Abschnitt ein eigenes Subjekt hat oder ob sich ein Abschnitt auf das Subjekt aus dem vorherigen Teil bezieht, weil sich dadurch die Position des Verbs im Satz verändert.

Anknüpfend sei die Identifikation des Nomens und dessen zugehöriger Teil als eines der Hauptaufgaben zu erwähnen. Wenn hier der Algorithmus scheitert, ist der Satz meistens kaum noch lesbar beziehungsweise schwer zu verstehen. Umso wichtiger ist es an dieser Stelle, sichere Regeln und Bedingungen festzulegen. Dazu benötigen wir nahezu alle Felder. Allerdings ist hauptsächlich das Feld „HEAD“ nicht korrekt oder „leer“. Durch eine etwas lockerere Regelführung versuchen wir die dem Nomen zugehörigen Wörter dennoch zu erkennen. Generell gehen wir rekursiv rückwärts den Abschnitt des Satzes durch und versuchen das am weitesten hinten stehende Wort, welches zum Nomen gehört, zu erkennen.

Ohne weitere Ausführung können wir an dieser Stelle noch erwähnen, dass die Benutzereingaben manchmal der Umgangssprache entsprechen. Ein Teilgebiet davon ist das Benutzen von Füllwörtern wie „ja“ oder „nur“, welche ebenfalls im Bezug auf die Platzierung des Verbs gesondert behandelt werden müssen.

Abschließend sei noch zu erwähnen, dass einige Sätze nicht verwertbar sind, da grobe Fehler vorliegen. Beispielsweise werden mehrere Sätze von D-BAS nicht als unterschiedliche Sätze erkannt und werden bei der Transformation als ein einzelner Satz behandelt oder Anführungszeichen werden nicht richtig von D-BAS verarbeitet und erscheinen bei der Ausgabe (unsere Eingabe) als „quote“ oder „& quote“. Diese Fälle werden in der Phase der Implementierung ignoriert, weil die Sätze nicht der vorgegebenen Form entsprechen oder falsch importiert werden.

3.3.3 Falsch umgewandelte Sätze im Trainingsset

Am Ende der Implementierung werden alle Sätze, welche verwertbar sind, bis auf 2 Ausnahmen erfolgreich umgewandelt. Diese schauen wir uns im Folgenden genauer an.

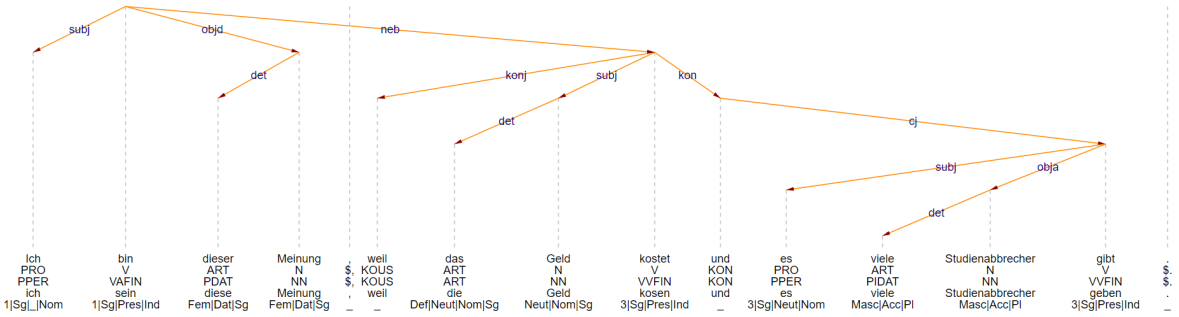

Abbildung 3.8: Grafik mit ParZu generiert [Sen+09]: Falsch erkannte Zugehörigkeit des Artikels und Subjekts in ParZu

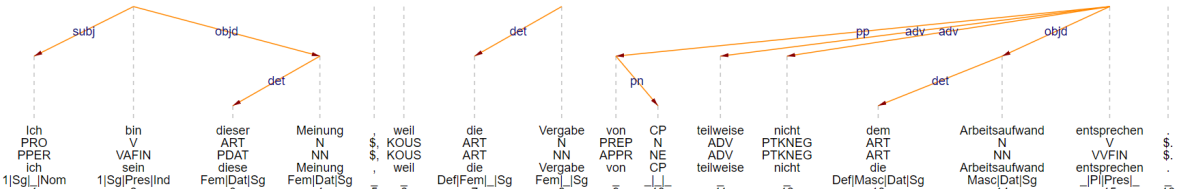

Abbildung 3.9: Grafik mit ParZu generiert [Sen+09]: Falsche erkannte Zugehörigkeit von „von“ bei falscher Benutzereingabe

In der Abbildung 3.8 und Abbildung 3.9 stellen die Pfeile die Abhängigkeiten innerhalb des Satzes dar („HEAD“). In der ersten Zeile sind die Wörter („FORM“), in der zweiten die universellen POS-Tags („UPOS“), in der dritten die sprachspezifischen POS-Tags („XPOS“), in der vierten Zeile das Lemma („LEMMA“) und in der letzten Zeile sind die syntaktischen und morphologischen Informationen („FEATS“) zu finden.

Bei der ersten Ausnahme (Abbildung 3.8) wird der Artikel „das“ fälschlicherweise zusammen mit dem Subjekt „Geld“ verbunden. Wir nehmen an, dass im Originalsatz „Geld“ allerdings ein Akkusativ ist. Diese Verbindung zwischen Artikel und Subjekt können wir nicht einfach, ohne Einfluss auf die anderen Sätze auszuüben, lösen. Der falsch transformierte Satz lautet „Das Geld kostet und es gibt viele Studienabbrecher.“, aber soll (vermutlich) „Das kostet Geld und es gibt viele Studienabbrecher.“ lauten.

In dem zweiten Satz (Abbildung 3.9) wird „CP“ nicht erkannt („FEATS“ ist leer) und das dazugehörige „von“ wird falsch interpretiert und zeigt nicht auf das Subjekt, wodurch das Verb missverständlich zwischen dem Subjekt und dem Zusatz gestellt wird. Allerdings ist die Benutzereingabe nicht korrekt. Wenn das Verb nicht den falschen Numerus besitzen würde, würde ParZu „vor“ richtig interpretieren und zum Nomen zuordnen. Allerdings ist der Eingabefehler nicht gravierend und die richtige Erkennung der Satzstruktur soll für den normalen

Einsatz auch mit kleineren Eingabefehlern funktionieren. Der falsch transformierte Satz lautet „Die Vergabe entsprechen von CP teilweise nicht dem Arbeitsaufwand.“, aber soll zu „Die Vergabe von CP entsprechen teilweise nicht dem Arbeitsaufwand.“ transformiert werden.

In Kapitel 3 haben wir das Grundkonzept beziehungsweise die Vorgehensweise des Algorithmus kennengelernt. Außerdem kennen wir nun die größten Schwierigkeiten bei der Umstellung eines Nebensatzes in einen Hauptsatz und haben einige individuelle Regeln kurz beleuchtet. Mit der Fertigstellung der Implementierung werden nun fast alle verwertbaren Sätze korrekt umgeformt und wir können in Kapitel 4 die aus dem Testset generierten Ergebnisse auswerten.

Kapitel 4

Evaluation

In Kapitel 4 werden die Ergebnisse aus dem Testset ausgewertet. Anfangs werden wir kurz auf die Auswertungsmetriken eingehen. Darauffolgend werden wir uns die Daten aus der Auswertung zum Testset genauer anschauen, untersuchen wie diese Zustande kommen und eventuell schon Verbesserungsvorschläge bezüglich des Algorithmus geben. Im Anschluss werden die daraus gewonnenen Ergebnisse mit den Ergebnissen aus der am Anfang erwähnten Arbeit verglichen. Abschließend werden wir auf weitere Problemstellungen eingehen und einen Lösungsansatz zum Filtern von Benutzereingaben vorstellen.

4.1 Auswertungsmetrik

Als Erstes stellt sich die Frage, welche Daten wir auswerten wollen. Da die Daten aus zwei Feldexperimenten stammen und der Algorithmus auch bei falschen Eingaben funktionieren soll, entscheiden wir uns dazu alle Datensätze auszuwerten. Wir werden sowohl die richtigen als auch die falschen Benutzereingaben evaluieren. Doppelte Einträge werden nicht bewertet. Wir kategorisieren die Benutzereingabe in zwei unterschiedliche Kategorien. Wir markieren entweder den Satz als richtig oder als falsch. Dies geschieht anhand eines Kompromisses zwischen der grammatikalischen Korrektheit und des menschlichen Verständnisses. Mit menschlichen Verständnis meinen wir, dass wir beurteilen, ob der Satz selbst mit kleineren grammatikalischen Fehlern verständlich ist. Wir haben uns für diese Vorgehensweise entschieden, weil wir alle Sätze korrekt umwandeln möchten. Wir müssen damit rechnen, dass Benutzereingaben nicht immer korrekt sind und meistens kleinere oder größere Fehler

vorhanden sind. Mit diesen Fehlern in der Eingabe muss unser Programm auch umgehen können, um dem Anspruch für ein geeignetes Tool gerecht zu werden. Demnach entscheiden wir die Korrektheit des Satzes nicht nur strikt anhand der Grammatik. In diesem Zusammenhang können wir einen richtigen Satz so interpretieren, dass dieser Allgemein anerkannt wird und wahrscheinlich im richtigen Einsatz auch vorkommt.

Ein weiteres Ziel hinter dieser Vorgehensweise ist es, realistisch einschätzen zu können, inwiefern der Algorithmus mit schlechten Eingaben zurecht kommt. Schlecht bedeutet in diesem Fall, dass man als fremder Benutzer Schwierigkeiten hat, den vorliegenden Satz zu verstehen oder wenn grobe Fehler in der Grammatik, Rechtschreibung, Semantik oder Syntax vorliegen. Wie bereits erwähnt, erfolgt dies manuell, allerdings werden die Bewertungen bezüglich der Richtigkeit der Daten der Feldexperimente entsprechend bei den Bewertungen der umgewandelten Sätze übernommen. Das heißt, falls wir bei der Benutzereingabe feststellen, dass zu viele Rechtschreibfehler vorhanden sind und wir diesen Satz als falsch werten, wir dies auch bei der Bewertung des umgewandelten Satzes übernehmen. Somit vermeiden wir sowohl „unechte“ Verbesserungen, als auch „unechte“ Verschlechterungen zwischen dem Originalsatz und dem umgewandelten Satz. Unabhängig davon kann es vorkommen, dass ein Satz im Original falsch ist, aber nach der Umwandlung richtig ist und umgekehrt. Der Wechsel von falsch nach richtig und von richtig nach falsch ist demnach unabhängig von den gemeinsamen Fehlern und bezieht sich hauptsächlich auf die Positionierung des Verbs.

Betrachten wir zum Beispiel folgenden Satz aus dem Testset: „Ich bin dieser Meinung, weil der Inhalt der Module ist für die theoretische Informatik sehr wichtig.“ Diese Benutzereingabe bewerten wir als inkorrekt, da das Verb im Satz falsch positioniert ist. Unser Algorithmus erzeugt folgende Umwandlung: „Der Inhalt der Module ist für die theoretische Informatik sehr wichtig.“ In diesem Satz ist das Verb an der richtigen Stelle und der Satz ist generell grammatikalisch korrekt. Somit bewerten wir die Transformation als korrekt.

Unabhängig davon beurteilen wir noch separat, ob die Umwandlung den Satz verständlicher gemacht, unverständlicher gemacht oder keinen Einfluss auf die Verständlichkeit hat. Diese Bewertung soll ein noch besseres Verständnis über die Güte des Algorithmus geben. Wenn wir einen Eingabesatz als falsch bewerten, können wir mit der oben genannten Methode keine vollständige Aussage über die Güte der Umwandlung machen. Wir können lediglich Verbesserungen erkennen, indem wir den umgewandelten Satz als richtig bewerten. Der Satz kann sich beim Umwandeln sowohl verschlechtert haben, als auch von der Verständlichkeit gleich geblieben sein, wenn wir in beiden Fällen den Satz als falsch bewerten. Analoges gilt auch

für den invertierten Fall. Die neue Metrik gibt genau an, in welchen solcher Fälle die Verständlichkeit gleich geblieben ist und in welchen sich diese verschlechtert beziehungsweise verbessert hat.

Betrachten wir beispielsweise folgenden Satz aus dem Testset: „Ich bin dieser Meinung, weil die Studierenden, die Mathe Informatik doppelt studieren sich dies selbst ausgesucht haben.“, welcher von unserem Algorithmus zu folgendem Satz umgewandelt wird: „Die Studierenden studieren , die Mathe Informatik doppelt sich dies selbst ausgesucht haben.“ Bei der Eingabe fehlt in diesem Beispiel das zweite Komma, um den Relativsatz „die Mathe Informatik doppelt studieren“ korrekt abzugrenzen. In der Transformation fehlt das Komma offensichtlich auch, da wir keine Satzzeichen hinzufügen oder entfernen. Mithilfe der ersten Metrik können wir nur feststellen, dass beide Sätze grammatikalisch falsch sind. Mit der zweiten Metrik können wir aber herausfiltern, dass die Transformation zu einem Satz führt, den man - wenn überhaupt - nur schwer verstehen kann. Zwar ist die Eingabe grammatikalisch inkorrekt, aber der Benutzer hat dennoch die Möglichkeit, den Satz zu verstehen. Dieses Verständnis ist beim transformierten Satz nicht mehr direkt gegeben. Diese Verschlechterung des Verständnisses können wir so mit der zweiten Metrik festhalten und untersuchen.

4.2 Auswertung des Testsets

4.2.1 Allgemeine Auswertung

In Abschnitt 4.2 werten wir die Ergebnisse aus dem Testset aus. Das Testset besteht aus 216 Sätzen. Davon werden wir 213 auswerten und die übrigen drei Eingaben als Duplikate ignorieren. Zuerst kategorisieren wir, wie oben erwähnt, die Benutzereingaben als richtig oder als falsch. Aus der Tabelle 4.1 können wir entnehmen, dass wir ungefähr 72 % der Eingaben als richtig bewerten. Wir fordern von unserem Algorithmus, dass alle richtigen Sätze korrekt umgewandelt werden. Allgemein haben wir dem Testset eine hohe Toleranzgrenze für Fehler eingeräumt, da spätestens in der richtigen Nutzung des Algorithmus diese Fehler wieder vorkommen können und wir diese Sätze ebenfalls richtig umwandeln möchten. Diesen Aspekt müssen wir im späteren Verlauf berücksichtigen, wenn wir im Folgenden die Umwandlungen der Sätze auswerten. Es kann demzufolge sein, dass alle richtigen Eingaben korrekt umgewandelt werden, aber ein paar der „leicht falschen“ Eingaben, welche als

Tabelle 4.1: Testset: Relative und absolute Anzahl an korrekten beziehungsweise falschen Benutzereingaben laut unserer Metrik

Art des Satzes	absolute Anzahl	relativer Anteil
korrekter Satz	153	71,83 %
falscher Satz	60	28,17 %

korrekt bewertet worden sind, nicht korrekt umgewandelt werden und somit die Ergebnisse negativ beeinflussen.

Eine Fehlerrate von ungefähr 28 % ist relativ hoch. Es ist anzunehmen, dass wir einige dieser Sätze nicht richtig umwandeln können. Wir bezeichnen solche Fehler als schwerwiegende Fehler. Von den 60 falschen Eingaben enthalten 29 Eingaben schwerwiegende Fehler und 31 Eingaben „normale“ Fehler. Schwerwiegende Fehler sehen wir als unlösbar beziehungsweise nur sehr schwer lösbar an. Solche Eingaben sind zum Beispiel falsch formatiert und enthalten Semikolons anstatt Leerzeichen, enthalten mehrere Sätze, enthalten zu viele/wenig Wörter oder sind schlichtweg nicht verständlich. Die anderen Fehler, welche wir als normale Fehler betiteln, sind nicht so schwerwiegend und manche davon können sogar aus subjektiver Sicht gerade noch zu den korrekten Sätzen gezählt werden. Durch die Metrik, in der wir die Verständlichkeit der Sätze separat bewerten, soll dieser Interpretationsfreiraum keinen Einfluss auf die Bewertung des Algorithmus haben. Dennoch müssen wir an dieser Stelle darauf hinweisen, dass wir nicht alle Sätze einzeln durchgehen können, um unsere Bewertungen zu begründen. Dafür gibt es zu viele Einzelfälle. Stattdessen erklären wir die Bewertung und Auswertung anhand des vorangestellten allgemeinen Bewertungsschemas, auf einige ausgewählte Sonderfälle und subjektiv festgelegte Entscheidungen, welche in den nächsten Abschnitten aufgeführt und begründet werden.

Anhand von der Tabelle 4.2 können wir feststellen, dass 143 Sätze korrekt umgewandelt werden. Das entspricht ungefähr 67 % und liegt knapp unter dem Anteil der korrekten Benutzereingaben (72 %). Somit können wir jetzt schon festlegen, dass nicht alle korrekten Benutzereingaben korrekt umgewandelt werden. Mit 70 falsch bewerteten Sätzen sind 10 Sätze mehr falsch umgewandelt worden als in der Eingabe als falsch bewertet. Im Folgenden analysieren wir genauer, wie diese Werte zustande kommen.

Tabelle 4.2: Testset: Relativer und absoluter Anteil an korrekten beziehungsweise falschen Transformationen laut unserer Metrik

Art der Umwandlung	absolute Anzahl	relativer Anteil
korrekte Umwandlung	143	67,14 %
falsche Umwandlung	70	32,86 %

4.2.2 Analyse richtiger Eingaben und falscher Transformationen

In der Tabelle 4.3 ist die Relation zwischen Eingabe und Umwandlung dargestellt. Im ersten Schritt betrachten wir die 22 Fälle, in denen ein korrekter Satz falsch umgewandelt wird und die zwölf Fälle, bei denen eine falsche Eingabe richtig umgewandelt wird. Mit der zweiten Metrik werden wir im nächsten Schritt die üblichen Sätze genauer analysieren, indem wir uns anschauen, inwieweit die Umformulierung Einfluss auf die Verständlichkeit hat.

Betrachten wir nun zuerst die 22 Fälle, bei denen eine korrekte Eingabe und eine falsche Umwandlung vorliegt. Dabei ist es möglich, dass mehrere Fehlertypen innerhalb eines Satzes vorkommen.

In fünf Fällen ist die Präfix-Erkennung beim Verb nicht korrekt. Im Algorithmus liegt eine vorher festgelegte Liste mit trennbaren Präfixen vor. Diese scheint nicht vollständig zu sein und auf dem ersten Blick auch falsche Präfixe zu enthalten. Allerdings gibt es auch Präfixe, welche kontextspezifisch sowohl trennbar als auch nicht trennbar sind. Dies kann der Algorithmus in dieser Form nicht identifizieren. Dieses Problem haben wir schon in der Implementierungsphase erkannt und in Kapitel 3 kurz beschrieben. Eine mögliche Lösung stellt eine Kontextanalyse dar, welche allerdings für jeden Fall separat betrachtet werden muss. Diese soll je nach Kontext entscheiden, ob das Präfix getrennt werden soll oder nicht. Der Zeitaufwand verhindert eine Umsetzung im Rahmen dieser Arbeit. Im folgenden Beispiel aus dem Testset wird das Präfix „statt“ nicht vom Verb getrennt, obwohl es offensichtlich abgetrennt werden muss: „So ziemlich stattfindet jede Informatik Vorlesung nur alle zwei Semester.“

In einem Satz wird der zweite Teil des Satzes durch eine spezifische Regel nicht umgewandelt. Die Regel besagt, dass, wenn nur Verben im zweiten Teil enthalten sind, wir den Satz nicht umwandeln. Die Regel ist in dieser Form nicht richtig und somit ist dort die Positionierung des Verbs falsch. Der konkrete Satz im Testset lautet „Ältere sowie verschmutzte Tas-

taturen sind unhygienisch und ausgetauscht werden sollten.“, aber das Verb „sollten“ muss nach vorne gestellt werden.

In einem weiteren Fall steht das Verb im originalen Teil alleine nach dem Komma und im umgewandelten Satz wird das Verb vor den Nebensatz platziert. Da das Verb in diesem Fall richtigerweise auch nicht getrennt wird, kommt hinter dem Komma des Nebensatzes nur der Punkt, aber das Komma wird von unserem Algorithmus nicht entfernt. Dies ist grammatikalisch inkorrekt und muss separat behandelt werden. Im Testset taucht dieser Fall im folgenden Satz auf: „Der Computerraum braucht im EG neue Hardware , insbesondere Peripherie „.“

Ein Sonderfall stellt auch der Satz „Ich bin dieser Meinung, weil die bisherigen drei Veranstaltungen zu sehr auf Mathematiker eingestellt und daher für Informatiker oft nicht anschaulich sind.“ dar. In diesem Fall wird der Satz durch das „und“ getrennt, was zu einer falschen Platzierung der Verben führt. An dieser Stelle müssen wir entweder ein „fehlendes“ sind im ersten Abschnitt einfügen oder den Satz nicht trennen, da wir das Verb im zweiten Abschnitt nach vorne ziehen können. Der falsch umgeformte Satz lautet wie folgt: „Die bisherigen drei Veranstaltungen eingestellt zu sehr auf Mathematiker und sind daher für Informatiker oft nicht anschaulich.“ Da ParZu den Satz nicht vollständig erkennt und zufällig genau für das Verb „eingestellt“ vor dem „und“ keine Abhängigkeit einträgt, greift unsere Regel in diesem Fall nicht. Normalerweise überprüfen wir, ob sich das Verb vor dem „und“ auf ein Wort hinter dem „und“ bezieht. Dies sollte in den obigen Beispiel das Hauptverb „sind“ sein. Falls wir diese Abhängigkeit vorliegen haben, soll der Satz an dieser Stelle nicht getrennt werden, was auch in diesem Beispiel richtig wäre. Allerdings trägt ParZu für das Verb keine Abhängigkeit ein, sodass wir den Satz nicht korrekt identifizieren können und falsch umwandeln. Eine Regel, welche überprüft, ob das Verb auf „0“ zeigt (also keine Abhängigkeit erkannt wird) und dann den Satz nicht trennt, kann nicht eingebaut werden, da sonst andere Sätze falsch transformiert werden. Um dieses Aufgabe zu lösen, ist eine genauere Untersuchung notwendig.

Manchmal bezieht sich ein Satzteil nach dem Subjekt auf das Subjekt selbst und das Verb muss dahinter platziert werden. In anderen Fällen wiederum muss das Verb zwischen dem Subjekt und dem anschließenden Satzteil platziert werden. In fünf Fällen ist der Teil, welcher zum Subjekt gehört, von ParZu beziehungsweise von unserem Programm falsch erkannt worden. Zum Einen ist fälschlicherweise zu viel dem Subjekt zugeordnet worden und zum Anderen ist zu wenig zum Subjekt zugeordnet worden. Somit ist die Platzierung des Verbs zum Einen hinter dem Objekt und zum Anderen vor dem Zusatz des Subjekts. Bei einem Satz

fehlt zudem noch die Kommasetzung vor einem Nebensatz. Beim Lesen der Benutzereingabe ist dies im Bezug auf das Verständnis des Satzes nicht negativ aufgefallen und deswegen wurde der Satz als richtig markiert. Im transformierten Teil führt das Fehlen der Kommata zu einer falschen Platzierung des Verbs und zu einem schlechteren Verständnis des Satzes. Ein Beispiel aus dem Testset für den Fall, dass zu viel zum Subjekt gezählt wird bietet nachstehender Satz: „Jene Module von den Meisten werden , die es bestehen , vergessen.“ Wohingegen der Satz „Zertifizierungen sind wie SCRUM-Master eine wichtige Errungenschaft für heutige Arbeitnehmer.“ ein passendes Beispiel für den anderen Fall darbietet.

In zwei Fällen reicht es nicht aus, das Verb zu tauschen. Ein Adverb, welches im Nebensatz zumindest umgangssprachlich korrekt platziert wurde, muss im Hauptsatz getauscht werden. Das ist im Algorithmus nicht implementiert und demnach ist der Satz falsch transformiert. Im zweiten Satz steht ein „sich“ vorne im Satz. Im Hauptsatz muss es hinter dem Verb platziert werden. Dies wird auch in der Umwandlung gemacht, allerdings wird das Verb hinter dem Subjekt platziert. Das ist an dieser Stelle falsch und die Regel im Algorithmus muss modifiziert werden. Im transformierten Satz „Informatiker trinken generell sehr gerne Kaffee , und wäre daher das gut für das Image der Informatik Fakultät.“ aus dem Testset muss das „das“ im zweiten Teil weiter nach vorne („und das wäre daher gut“) gesetzt werden.

In vier Fällen ist die Positionierung des Verbs im Bezug auf das Adverb falsch. In drei Fällen ist das Verb fälschlicherweise hinter dem Adverb und in einem Fall vor dem Adverb platziert worden. In Kapitel 3 haben wir die Schwierigkeit dieses Problems kurz angesprochen. Die bereits vorhandenen Regeln bezüglich der Behandlung von Adverbien müssen zum Einen überprüft und zum Anderen erweitert werden. Im folgenden Beispiel muss das Verb „ist“ vor dem Adverb „lediglich“ positioniert werden: „Es hat keine Auswirkung auf die Qualität des Studienganges , sondern lediglich ist eine kleine Subventionierung von Studierenden.“

Ein ähnliches Problem tritt bei der Erkennung des Subjekts nach dem ersten Teil des Hauptsatzes auf. Es liegen drei Fälle vor, in denen das Subjekt nicht erkannt oder ein Objekt fälschlicherweise als Subjekt erkannt wird. Dies liegt in zwei Fällen daran, dass das Wort an sich (zum Beispiel „Propra1-Denken“) nicht erkannt wird. Die Regeln müssen an dieser Stelle ebenfalls angepasst werden. Ein Beispiel für dieses Phänomen ist der Satz „Viele Ressourcen sind nötig , um ein solches Leihsystem zu verwalten und kosten Mehrfachsteckdosen / Verlängerungskabel wenige Euro und können nötigenfalls selbst mitgebracht werden.“ Dort wird „Mehrfachsteckdosen / Verlängerungskabel“ nicht als Subjekt des Abschnittes erkannt.

Anhand von drei Fällen können wir feststellen, dass der Algorithmus momentan noch die Platzierung des Verbs zwischen Sonderzeichen, wie zum Beispiel Klammern und Anführungszeichen, erlaubt. Wenn wir uns die Implementierung genauer anschauen, können wir den Fehler genauer bestimmen. Während der Umwandlung des Satzes wird das Verb neu positioniert. Bevor wir das Verb in den neuen Satz einfügen, werden einige Eigenschaften überprüft.

Eine davon ist, ob vor dem Verb im neuen Hauptsatz ein Satzzeichen steht. Im Trainingsset gibt es Eingabesätze, bei denen das Verb nach einem Komma positioniert ist. Bei der Umformung soll das Verb im neuen Hauptsatz vor dem Komma positioniert werden. Beispielsweise wird „Ich bin dieser Meinung, weil man, um auf dem aktuellen Stand bleiben zu können, täglich sehr viele Seiten besuchen muss.“ zu „Man muss , um auf dem aktuellen Stand bleiben zu können , täglich sehr viele Seiten besuchen.“ Dadurch können wir pauschal nicht festlegen, dass das Verb hinter dem Satzzeichen positioniert wird. Auf der anderen Seite können wir analog auch nicht pauschal festlegen, dass das Verb immer vor dem Satzzeichen positioniert wird. Zum Beispiel wird „Ich bin dieser Meinung, weil die Studenten, die nicht abbrechen, dennoch zu wenig Ahnung haben, um effektiv in das Berufsleben zu starten.“ zu „ Die Studenten , die nicht abbrechen , haben dennoch zu wenig Ahnung , um effektiv in das Berufsleben zu starten.“ Deswegen werden in dem Algorithmus die Vorkommnisse des spezifischen Satzzeichens vor der vermeintlich neuen Position des Verbs gezählt und bei einer geraden Anzahl wird das Verb hinter dem Satzzeichen und bei einer ungeraden Zahl vor dem Satzzeichen platziert.

Während der Implementierungsphase mit dem Trainingsset sind nur Kommata von dieser Regel betroffen, weswegen nicht aufgefallen ist, dass noch eine kleine Ergänzung für andere Satzzeichen wie Klammern oder Anführungszeichen eingefügt werden müssen. Im Falle der Klammer „)“ wird zum Beispiel nur dieses Symbol im Satz gesucht und nicht noch zusätzlich „(“, weswegen die Anzahl nicht korrekt berechnet wird und somit das Verb falsch positioniert wird. Eine Regel, welche dies verhindert, ist leicht zu implementieren und sollte noch eingefügt werden.

Wir haben uns alle Fälle angeschaut, in denen ein richtiger Satz falsch umgewandelt wird. Wir können festhalten, dass einige Fehler durch die falsche Erkennung durch ParZu zustande kommen. Ein anderer Teil kommt durch die nicht korrekten beziehungsweise unvollständigen Umformungs-Regeln in der Implementierung zustande. Der Großteil kann aber durch die Anpassungen der Regeln (zum Teil leicht) korrekt umgeformt werden.

4.2.3 Analyse falscher Eingaben und richtiger Transformationen

Als Nächstes analysieren wir die von uns als falsch markierten Benutzereingaben, welche korrekt transformiert werden. Am Ende der Bewertung hat sich gezeigt, dass in allen zwölf Fällen der gleiche Fehler vorliegt. In diesen Sätzen wird das Verb wie in einem Hauptsatz und nicht am Ende des Satzes beziehungsweise Abschnitts platziert. Die Umwandlungen sind in diesen Fällen alle korrekt, wenn auch oft am Satz an sich nichts verändert wird, da diese schon in der richtigen Form vorliegen. An dieser Stelle können wir auch festhalten, dass der Fehler nicht gravierend ist und die Eingabe als richtig markiert werden kann. Dadurch verändern wir den Anteil an falschen Umwandlungen nicht. Wir können aber feststellen, dass die Sätze in der umgewandelten Form deutlich angenehmer zu lesen sind beziehungsweise einem das Verständnis des Inhalts leichter fällt, sodass wir uns am Anfang dafür entschieden haben, diese Sätze als inkorrekt zu markieren.

4.2.4 Verbesserungen/Verschlechterungen des Verständnisses von Sätzen

Überleitend dazu können wir diejenigen Fälle begutachten, welche sowohl eine korrekte Eingabe als auch eine korrekte Übersetzung haben. Analog gilt dasselbe für falsche Eingaben und falsche Übersetzungen. Im Zusammenhang mit unserer zweiten Metrik, welche die Verständlichkeit des umformulierten Satzes im Bezug auf die Eingabe beurteilt, können wir die Sätze aufdecken, welche sich entweder „verbessert“ oder „verschlechtert“ haben.

Zunächst betrachten wir die Verschlechterungen. Es gibt im Testset genau 32 Fälle, in denen wir Verschlechterungen feststellen können, wovon wir in genau dreizehn Fällen entweder die Eingabe und die Umformung als korrekt oder falsch bewerten. Bei genauer Betrachtung fällt auf, dass in allen Fällen die Eingabe und die Umformung falsch ist. Das heißt, dass wir keine Verschlechterung erzeugen, wenn sowohl die Eingabe als auch die Umwandlung korrekt ist. Wir können also davon ausgehen, dass diese Verschlechterungen auf die falsche Eingabe zurückzuführen ist und dies kein direkter Fehler im Algorithmus ist. In den meisten Fällen fehlt die Kommasetzung oder das Komma ist falsch platziert. Es ist keine Überraschung, dass dies zu Fehlern führt, da im ersten Schritt des Algorithmus anhand dieser Kommata der Satz in Abschnitte unterteilt wird. Wenn diese Kernfunktion wegen des Fehlens oder falschen Platzierens von Kommata nicht funktioniert, kann der Algorithmus den Satz nicht richtig

Tabelle 4.3: Testset: Absolute Anzahl zwischen den Relationen von korrekten/falschen Sätzen zu Transformationen

	korrekte Transformation	falsche Transformation
korrekter Satz	131	22
falscher Satz	12	48

transformieren.

Wie zu erwarten, finden wir keine Fälle, wo die Transformation als korrekt bewertet wird und wir in der zweiten Metrik denselben Satz als Verschlechterung einstufen. Demzufolge sind die üblichen 19 Fälle eine Teilmenge der ersten Metrik im Falle einer korrekten Eingabe und einer falschen Transformation. Dies heißt lediglich, dass in 19 von den 22 oben erwähnten Vorkommnissen die falsche Übersetzung zu einer starken Verschlechterung des Verständnisses des Satzes führt. Die verschiedenen Arten der Fehler wurden vorhin schon genauer erläutert.

In Abschnitt 4.2 haben wir die Ergebnisse aus dem Testset ausgewertet, uns die Fehler und Fehlerquellen genauer angeschaut und zum Teil schon Verbesserungsvorschläge gegeben. Die hohe Übereinstimmung unserer beiden Bewertungsmetriken an dieser Stelle zeigt, dass die subjektive Bewertung beider Metriken deckungsgleich ist. Zudem konnten wir so feststellen, welche Sätze nach der Transformation qualitativ schlechter beziehungsweise besser geworden sind. Als nächstes vergleichen wir die Ergebnisse aus dieser Arbeit mit den Ergebnissen von Borodkin [Bor20].

4.3 Vergleich zur anderen Arbeit

In Abschnitt 4.3 vergleichen wir die Ergebnisse aus den vorherigen Abschnitten mit den Ergebnissen aus der Arbeit von Kirill Borodkin [Bor20]. In dieser Arbeit werden die Nebensätze aus D-BAS mit einer anderen Methode umgewandelt. Zuerst werden die Nebensätze in die englische Sprache übersetzt. Da die Satzstruktur von Neben- und Hauptsätzen im Englischen gleich ist, soll auf diese Weise eine Hauptsatzstruktur generiert werden. Diese „englischen Hauptsätze“ werden dann anschließend zurück ins Deutsche übersetzt. Beim späteren Vergleich benutzen wir zum Einen den Begriff „Transformation“, um auf das Vorgehen in dieser

Tabelle 4.4: Testset: Relativer und absoluter Anteil der vier Kategorien aus der Arbeit von Borodkin [Bor20]

Art der Umwandlung	absolute Anzahl	relativer Anteil
gut	86	40,38 %
mittel	17	7,99 %
schlecht	36	16,9 %
nicht bewertet	74	37,74 %

Arbeit zu referenzieren, und zum Anderen den Begriff „Übersetzung“ für die Arbeit von Borodkin, um die Referenz auf beiden Werken einfacher unterscheiden zu können. Zudem sind in der Arbeit von Borodkin zwei unterschiedliche Eingabeformate für die Übersetzung gewählt worden. In der ersten Vorgehensweise wird der Nebensatz ohne Zusatz konvertiert und im zweiten Fall wird ein Hauptsatz („Ich bin dieser Meinung, weil“) vorangestellt. Dann wird der komplette Satz konvertiert und im Anschluss der Nebensatz extrahiert. Die Ergebnisse aus der Arbeit zeigen, dass die zweite Methode etwas bessere Ergebnisse erzielt. Deswegen ziehen wir, als Vergleich zu unseren Ergebnissen, diese Vorgehensweise für die Auswertung heran.

4.3.1 Anpassung der Bewertungsmetrik

Die Bewertung der Übersetzung in der Arbeit unterscheidet sich von der Bewertung in diesem Werk. Dort werden die Sätze in vier Kategorien unterteilt. Die erste Kategorie „gut“ beinhaltet alle Sätze, welche richtig umgewandelt werden. Die zweite Kategorie „mittel“ besteht aus den Sätzen, welche mittelmäßig gut umgewandelt werden. Falsch übersetzte Sätze werden zur dritten Kategorie „schlecht“ gezählt. Benutzereingaben, welche als falsch angesehen werden, werden nicht umgewandelt und fallen in die letzte Kategorie „nicht bewertet“. Die Ergebnisse aus der Arbeit sind in der Tabelle 4.4 aufgeführt.

Diese Art der Bewertung ist stark unterschiedlich zu der Bewertung, welche wir in dieser Arbeit gewählt haben. Um die Ergebnisse sinnvoll vergleichen zu können, müssen wir einige Anpassungen an dieser Bewertung vornehmen. Als Erstes lösen wir die letzte Kategorie „nicht bewertet“ auf, indem wir die Sätze jeweils in die Kategorien „gut“ oder „schlecht“ einordnen. Da wir in unserer Arbeit alle Sätze bewerten, übernehmen wir dieses Vorgehen auch in der anderen Arbeit, um die Güte der Übersetzung besser festlegen zu können. Wir

Tabelle 4.5: Testset: An unserer Metrik angepasster relativer und absoluter Anteil der guten/schlechten Übersetzungen aus der Arbeit von Borodkin [Bor20]

Art der Umwandlung	absolute Anzahl	relativer Anteil
gut	86	40,38 %
schlecht	127	59,62 %

haben dieses Vorgehen schon am Anfang von Kapitel 4 erläutert.

Zweitens lösen wir die Kategorie „mittel“ auf, indem ebenfalls solche Sätze entweder in die Kategorie „gut“ oder „schlecht“ eingeordnet werden. Der Grund hierfür ist, dass wir in unserer Auswertung eine solche Kategorie nicht benutzen und wir so eine bessere Vergleichsbasis schaffen. Außerdem ist die Bewertung und Einordnung der Sätze in Kategorien subjektiv und je mehr Kategorien wir schaffen, desto mehr Abweichungen erhalten wir in den subjektiven Bewertungen. Von den insgesamt 17 Sätzen, welche der Kategorie „mittel“ zugeordnet werden, ordnen wir drei der Kategorie „gut“ und 15 der Kategorie „schlecht“ zu. Die Aufteilung der Sätze geht eindeutig in die „negative“ Richtung. Das heißt, dass wir den Großteil der als „mittel“ bewerteten Sätze als falsch beurteilen. Da die Bewertung in beiden Fällen unterschiedlich und subjektiv erfolgt, können wir nicht festlegen, inwiefern dies den Vergleich beeinträchtigt. Zum Beispiel ist aufgefallen, dass in der Arbeit von Borodkin die Korrektheit des Satzes an sich nicht bewertet wird, sondern ob der übersetzte Hauptsatz dem originalen Nebensatz entspricht. Beispielsweise können wir folgende Eingabe aus D-BAS betrachten, welche in der Umwandlung um einen Hauptsatz ergänzt wird: „Ich bin der Meinung, weil Für Projekte bei den wir dann billig Gehäuse und selbst entwickelte Teile drucken können.“ Es ist offensichtlich, dass diese Eingabe laut unserer Metrik nicht korrekt ist. Der kalkulierte Hauptsatz mithilfe der Vorgehensweise von Borodkin lautet wie folgt: „Für Projekte, bei denen wir dann billige Gehäuse und selbst entwickelte Teile drucken können.“ Wie erwartet ist auch der Hauptsatz formal nicht korrekt, allerdings wird diese Umwandlung in der Bewertung von Borodkin in die Kategorie „gut“ eingestuft. In dieser Arbeit ordnen wir laut unserer definierten Metrik den Satz als „falsch“ ein. Aus diesem Grund und wegen dem deutlichen „negativen“ Trend bei der Verschmelzung der Kategorien bewerten wir die Ergebnisse aus der Arbeit von Kirill Borodkin nach unserer Metrik neu. Die Ergebnisse können wir der Tabelle 4.5 entnehmen.

Tabelle 4.6: Testset: Vergleich der als gut oder schlecht bewerteten Sätze aus der Arbeit von Borodkin nach unterschiedlichen Metriken

	korrekte Übersetzung	falsche Übersetzung
korrekte Umwandlung	63	2
falsche Umwandlung	23	34

4.3.2 Vergleich der subjektiven Bewertungen

Bevor wir zu der Analyse der Ergebnisse kommen, betrachten wir die Übereinstimmung beziehungsweise Abweichung der Bewertung zwischen den beiden Arbeiten. Um einen möglichst aussagekräftigen Vergleich ziehen zu können, bewerten wir nur die Sätze, welche von Borodkin entweder als „gut“ oder als „schlecht“ bewertet werden. In der Tabelle 4.6 vergleichen wir die Ergebnisse, der als „gut“ oder „schlecht“ bewerteten Sätze, von Borodkin mit unserer Metrik. Dabei geben die Zeilen „Umwandlung“ unsere Bewertung an und die Spalten „Übersetzung“ die Bewertung von Borodkin an. Knapp 20,5 % der betrachteten 122 Sätze werden anders eingestuft. Davon sind fast alle (mit Ausnahme von zwei Fällen) in unserer Arbeit als falsch eingestuft und in der Arbeit von Borodkin als richtig eingestuft. Demzufolge führen die Metriken in dieser Arbeit zu einer strengeren Bewertung als die Bewertungsmetrik aus dem anderen Werk. Zum Beispiel bewerten wir hier die Übersetzung von „Man“ zu „Sie“ als falsch, da je nach Kontext die Bedeutung abweichen kann. Beispielsweise wird „Ich bin der Meinung, weil man nicht ohne Bildschirme arbeiten kann“ zu „Sie können nicht ohne Bildschirme arbeiten.“ Ohne den Originalteil zu kennen, ist es nicht eindeutig, wie „Sie“ in diesem Fall interpretiert werden soll. In der Arbeit von Borodkin hingegen werden diese Sätze unterschiedlich und vermutlich kontextabhängig bewertet.

Dennoch werden in dieser Arbeit Sätze als richtig bewertet, obwohl diese in Borodkin als falsch gewertet werden. Beispielsweise bewerten wir die Übersetzung von „Student“ im originalen Text zu „Schüler“ im umgewandelten Text nicht als Fehler. Wir gehen davon aus, dass der Benutzer weiß, was mit „Schüler“ gemeint ist, und keine Schwierigkeiten hat den Satz zu interpretieren. Bei Borodkin hingegen wird diese Übersetzung als falsch eingestuft. Beide Bewertungen sind je nach Begründung vertretbar. Es kommen noch weitere unterschiedliche Bewertungen hinzu, die wir an dieser Stelle nicht konkret aufführen, da diese Einzelfälle zu unübersichtlich werden und wir nicht immer sicher sagen können, wie bei Borodkin die dortigen Bewertungen zu Stande gekommen sind. Für zwei unterschiedlich und komplett subjektive Bewertungen ist die Abweichung nachvollziehbar. Um die Güte der Transforma-

Tabelle 4.7: Testset: Absolute Anzahl der korrekt/falsch umgeformten Hauptsätze mithilfe von ParZu (Transformation) und der Methode von Borodkin (Übersetzung) nach unserer Metrik

	korrekte Übersetzung	falsche Übersetzung
korrekte Transformation	68	75
falsche Transformation	18	52

tion aber weiter beurteilen zu können, werden wir demnach unsere Metrik auf die Ergebnisse der Umwandlung von Borodkin anwenden.

4.3.3 Vergleich der Ergebnisse

Insgesamt werden laut unserer Metrik ungefähr 40 % der Sätze durch die Methode von Borodkin korrekt übersetzt. Tatsächlich ist die Anzahl an „guten“ Umwandlungen laut Borodkin identisch. Anzumerken ist, dass diese 86 Sätze nicht identisch zu unseren, als korrekt eingestuften, Sätzen sind. Viel mehr können wir der Tabelle 4.7 entnehmen, dass dort 18 Sätze korrekt übersetzt werden, aber wir mit unserem Algorithmus diese falsch transformiert haben. Auf der anderen Seite werden von unserem Algorithmus 75 (ungefähr 60 %) der falsch übersetzten Nebensätze richtig transformiert. Wenn wir die relativen Werte vergleichen, können wir feststellen, dass unser Verfahren deutlich bessere Ergebnisse bezüglich der korrekten Umwandlung liefert. Unser Algorithmus wandelt von allen Sätzen circa 67 % korrekt um und im Gegensatz dazu werden nach unserem Bewertungsschema circa 40 % der Sätze mithilfe der Methode von Borodkin korrekt umgewandelt. Das ist eine Verbesserung von über 50 %. Die von unserem Algorithmus erreichte korrekte Umwandlungsrate ist zwar deutlich besser als die Werte aus der Arbeit von Borodkin, allerdings reichen diese für einen praktischen Einsatz noch nicht aus. Der Anspruch liegt bei einer korrekten Umwandlung von fast allen Sätzen. Um diese Rate zu erreichen, müssen viel mehr Sätze betrachtet werden und mehr Regeln manuell eingerichtet werden. Eine angemessene Trainingssetgröße liegt bei ungefähr 1000 Sätzen und die entsprechende Testsetgröße bei ungefähr 2000 Sätzen, wie es zum Beispiel beim Erstellen des Dependency Parsers ParZu der Fall ist [Sen+09]. Der hohe Aufwand ist im Rahmen dieser Arbeit nicht möglich.

Wie vorhin schon kurz erwähnt, werden einige Sätze, welche von uns falsch transformiert werden, von Borodkin korrekt umgewandelt. Demzufolge könnte man versuchen, beide Me-

thoden miteinander zu verknüpfen. Dies ist keine leichte Aufgabe, da wir zuerst automatisch feststellen müssen, welche Sätze korrekt transformiert werden und welche falsch transformiert werden. Hinzu kommt noch die Herausforderung, solche Sätze korrekt umzuformen, wenn beide Ansätze diese nicht korrekt verarbeiten können.

Ein weiterer Ansatz zur Verbesserung der korrekten Umwandlungsrate ist die Erkennung von Sätzen, welche wir nicht korrekt transformieren können. Die Sätze aus der Benutzereingabe können herausgefiltert und nicht umgewandelt werden, um eine Verschlechterung des Verständnisses vorzubeugen. Dieser Ansatz wird in Abschnitt 4.4 dieses Kapitels aufgeführt. Aus zeitlichen Gründen werden wir einen primitiven Ansatz nur testen, aber nicht weiterentwickeln können.

4.4 Ansatz zum Filtern der Benutzereingaben

4.4.1 Implementierung des Filters

Ein alternativer Ansatz zur Umwandlung aller Benutzereingaben ist die Umwandlung der korrekten Eingaben beziehungsweise solcher, welche von ParZu vermeintlich korrekt erkannt werden. Alle üblichen Eingaben sollen in der originalen Form weitergegeben werden. Solche Sätze werden folglich nicht umgewandelt und sind sehr wahrscheinlich nicht vollkommen richtig, aber eine Umwandlung von solchen Sätzen kann (sehr wahrscheinlich) eine Verschlechterung des Verständnisses verursachen. Das heißt, dass wir in diesen Fällen lieber die falschen Eingaben übernehmen und das Risiko einer Verschlechterung nicht eingehen. Das Ziel und die Vorgabe hinter dieser Vorgehensweise ist es, alle betrachteten Sätze korrekt umzuwandeln. Weiterhin bedeutet dies, dass der benötigte Aufwand extrem reduziert wird, da man falsche Benutzereingaben und Eingaben, welche ParZu nicht korrekt erkennt, nicht behandeln muss. Ein Nachteil ist offensichtlich, dass nicht alle Sätze betrachtet und somit nicht alle Sätze umgewandelt werden. Es stellt sich die Frage, wie man solche Sätze unterscheiden kann. Ein einfacher direkter Ansatz ist es, die Umwandlung von ParZu genauer zu betrachten. Wir überprüfen den Satz auf Wörter, welche keine Satzzeichen sind und im Feld „HEAD“ von ParZu eine 0 enthalten. Dies deutet darauf hin, dass ParZu den Zusammenhang der Wörter im Satz nicht richtig erkennt und somit das Dependency Parsing von diesem Satz nicht korrekt funktioniert. Diese Methode ist keineswegs vollständig. Wir können noch

viel mehr Eigenschaften überprüfen, um festzustellen, ob der Satz korrekt erkannt wird. Zum Beispiel wäre ein Abgleich des Numerus von dem Verb und Subjekt denkbar. Wie vorhin schon erwähnt, wäre der Aufwand zur vernünftigen Validierung dieses erweiterten Filters zu hoch.

4.4.2 Analyse des Trainingssets

Das Trainingsset wird nahezu komplett richtig transformiert. Dies bedeutet allerdings, dass wir anhand dieser Datenmenge nicht testen können, inwiefern der Ansatz funktioniert. Es werden eben auch (fast) alle Eingaben korrekt umgewandelt, welche Wörter enthalten, die im Feld „HEAD“ eine 0 enthalten. Demzufolge ziehen wir für das Testen dieser Funktion auch das Testset heran. Das ist ein weiterer Grund, warum wir diesen Ansatz nicht weiterentwickeln. Falls wir das Testset zum Trainieren des Algorithmus benutzen, können wir den Ansatz nicht evaluieren. Das jetzige Testset wird so zum Trainingsset umgestaltet und somit kann diese Datenmenge nicht mehr für die Evaluation des Filters benutzt werden.

In Kapitel 3 haben wir den Algorithmus soweit entwickelt, dass nur zwei Sätze nicht richtig umgewandelt werden. Diese betrachten wir als Erstes. In der Abbildung 3.9 wird der Satz als „falsch“ erkannt und somit im modifizierten Algorithmus nicht umgewandelt. In diesem Fall stimmt die Erkennung des Filters mit unserem Ergebnis aus Kapitel 3 überein. Zwar ist der Satz laut unserer ursprünglichen Metrik und grammatikalisch richtig, aber wir können ihn nicht korrekt verarbeiten. Der Satz wird richtigerweise als falsch markiert und nicht umgewandelt.

Der Satz aus der anderen Abbildung 3.8 wird von unserer modifizierten Implementierung als korrekt angesehen. Dieser Satz ist tatsächlich formal und inhaltlich korrekt, aber wir können den Satz nicht korrekt transformieren. Dementsprechend hat der Filter an dieser Stelle den Satz fälschlicherweise als korrekt markiert. Um herauszufinden, ob dies ein Einzelfall ist, betrachten wir als nächstes das Testset.

Tabelle 4.8: Testset: Absolute Anzahl der von unserem Filter als richtig/falsch bewerteten Eingaben in Relation zu den Transformationen

	korrekte Transformation	falsche Transformation
Eingabe als richtig bewertet	120	29
Eingabe als falsch bewertet	23	41

4.4.3 Analyse des Testsets

Aus der Tabelle 4.8 können wir entnehmen, dass für 161 Sätze (75,59 %) der einfache Ansatz korrekt funktioniert. In 23 Fällen (10,8 %) wird die Eingabe als falsch bewertet, die Transformation aber als korrekt. Dies ist an sich falsch, aber die Konsequenzen sind nicht schwerwiegend. Diese Sätze werden lediglich nicht umgewandelt, obwohl wir dies mit unserem Algorithmus korrekt durchführen können. Falls die Eingaben grammatikalisch korrekt sind, also wir mit dem Filter eine falsche Einschätzung erhalten, sind diese auch nach der Transformation korrekt, da wir diese Sätze nicht verändern. Das ist zwar ungünstig, aber nicht tragisch. Um zu überprüfen, ob falsche Benutzereingaben an sich richtig konvertiert werden können, aber durch den Filter nicht transformiert werden würden, untersuchen wir diese 23 Sätze und verknüpfen die Ergebnisse mit den Ergebnissen aus dem Abschnitt 4.2. Zudem untersuchen wir, warum der Filter diese Sätze als eine unkorrekte Eingabe markiert hat.

Eingabe als falsch bewertet und korrekt transformiert

Insgesamt sind sechs von den 23 Sätzen grammatikalisch falsch und werden somit korrekterweise von unserem Filter als falsch markiert. Allerdings kann unser Algorithmus diese richtig konvertieren, weswegen in der Theorie sechs richtige Umwandlungen wegfallen würden. In den meisten Fällen fehlt mindestens ein Komma. Aber in allen Fällen wird der letzte Teil von ParZu nicht vollständig erkannt. Damit meinen wir, dass nicht jedes Wort einen „Header“ ungleich 0 zugeordnet bekommt. Dies liegt entweder daran, dass entweder das Komma fehlt oder der Nebensatz grammatikalisch nicht korrekt ist.

Betrachten wir nun die üblichen Fälle, welche wir zumindest laut unserer Metrik als korrekte Benutzereingabe werten können. Folgender Satz wird nicht erkannt: „Ich bin dieser Meinung, weil das keine 10000€ Wert ist.“ Nach genaueren Untersuchungen und weiterem

Testen ist aufgefallen, dass das Euro-Symbol „€“ nicht als solches erkannt wird, sondern als substantiviertes Adjektiv. Dies ist falsch und kann entweder in ParZu behoben werden oder eine Regel zum Ausschreiben von Abkürzungen könnte um Währungszeichen erweitert werden.

Anknüpfend daran kann der als falsch erkannte Satz „Ich bin dieser Meinung, weil die Rechner stürzen regelmäßig ab bzw frieren ein.“ als richtig erkannt werden, wenn die Abkürzung ausgeschrieben wird.

In zwei Fällen sorgt ein Doppelpunkt im Satz dafür, dass nicht alle Satzteile miteinander verknüpft werden. Dies ist an sich nicht falsch, da durch ein Doppelpunkt der Satz klar getrennt wird. Im primitiven Ansatz kann eine Abfrage nach einem Doppelpunkt implementiert werden. Zusätzlich muss dann allerdings eine neue Regel eingebaut werden, welche andere Kriterien überprüft, weil sonst alle Sätze mit einem Doppelpunkt als richtig angesehen werden.

In fünf Fällen wird der jeweilige letzte Teil im Satz nicht richtig erkannt und mit dem vorherigen Abschnitt nicht verbunden. Woran das genau liegt, können wir nicht zu 100 % festlegen, aber wir vermuten, dass die Sätze teilweise zu verschachtelt sind oder ParZu mit den Satzzeichen (wie zum Beispiel Klammern) nicht zurecht kommt. Beispielsweise wird der Einschub in den Klammern im folgenden Satz nicht zum restlichen Satz in Verbindung gebracht: „Ich bin dieser Meinung, weil die ehemaligen Mathe-Pflichtmodule dann als Mathe-Nebenfach belegt werden könnten und dieses dadurch einfacher wird (die alten Pflichtmodule decken die 30CP sogar ganz ab).“

Eigennamen führen in zwei weiteren Sätzen dazu, dass ParZu die Abhängigkeiten nicht im kompletten Satz zuordnen kann. Beispielsweise werden im folgenden Satz die Modulnamen „Analysis I, II oder Lineare Algebra I“ nicht erkannt: „Ich bin dieser Meinung, weil dann Veranstaltung wie Analysis I, II oder Lineare Algebra I wegen sinkender Teilnehmerzahlen möglicherweise nur noch alle zwei Semester angeboten werden.“

In drei Fällen scheint es Probleme mit den Verben zu geben. Im ersten Fall werden die letzten zwei Verben am Schluss nicht zugeordnet („Ich bin dieser Meinung, weil dafür keine Fördergelder ausgegeben werden müssen, sondern dies im Rahmen einer Bachelorarbeit erledigt werden kann.“). Im zweiten Fall wird das letzte und einzige Verb am Ende nicht zugeordnet. Der Rest ist aber korrekt verarbeitet worden. Warum in diesem Satz „Ich bin dieser Meinung,

weil Akademiker meist als Führungspersonen eingestellt werden, wofür Zertifizierungen notwendig sind.“ nur das Verb nicht erkannt wird, können wir an dieser Stelle nicht erkennen. Im letzten Fall „Ich bin dieser Meinung, weil Rechnernetze einen eigenen Praktikumsraum hat.“ scheint ParZu den Numerus von „Rechnernetze“ als Plural zu erkennen, obwohl damit das Modul an sich, also Singular, gemeint ist. Somit bekommt ParZu Probleme, das Verb „hat“ dem Subjekt zuzuordnen. Es ist in der Tat so, dass, wenn wir das Subjekt zu „Rechnernetz“ ändern, ParZu die Abhängigkeiten im Satz korrekt erkennt.

Die übrigen Fälle sind besonders interessant. Dort wird der letzte Teil des Nebensatzes nicht erkannt, aber zusätzlich ist es auffällig, dass die Einschübe nach dem Verb vorkommen. Betrachten wir beispielsweise folgenden Satz: „Ich bin dieser Meinung, weil wir Informatik Studenten sind und keine Architektur Studenten.“ Der letzte Teil „und keine Architektur Studenten“ ist in dieser Form nicht ganz korrekt. Theoretisch müsste davor ein Komma platziert und das „und“ entfernt werden. Der Satz an sich ist so aber verständlich und wird von uns anhand unserer Metrik als korrekt angesehen. Wenn wir nun den Nebensatz vor das Verb einfügen und die entsprechenden Änderungen vornehmen, erhalten wir den Satz „Ich bin dieser Meinung, weil wir Informatik Studenten, keine Architektur Studenten, sind.“, welcher von unserem Filter als korrekt angesehen wird.

Gleiches können wir beim Beispiel „Ich bin dieser Meinung, weil 3D Drucker nicht nur von Architekturstudenten genutzt werden, sondern auch zur Herstellung von Reparaturteilen.“ beobachten. Wenn wir hier den letzten Nebensatz vor das Verb platzieren, erhalten wir folgenden Satz: „Ich bin dieser Meinung, weil 3D Drucker nicht nur von Architekturstudenten, sondern auch zur Herstellung von Reparaturteilen, genutzt werden.“ ParZu kann in diesem Satz allen Wörtern Abhängigkeiten zuordnen, wohingegen ParZu dies im originalen Satz nicht kann.

Das nächste Beispiel zeigt ein ähnliches Verhalten. Im folgenden Satz ist der letzte Nebensatz beziehungsweise Einschub hinter das Hauptverb (im transformierten Satz) platziert worden: „Ich bin dieser Meinung, weil die Abgabe der Lösungen nicht ansatzweise so viel Zeit / Komplexität erfordert, wie das Erarbeiten der Lösungen.“ Wenn wir nun, wie bei den anderen beiden Fällen, den Zusatz vor das Verb ziehen und die Kommata entfernen, erkennt ParZu die Zugehörigkeiten im Satz („Ich bin dieser Meinung, weil die Abgabe der Lösungen nicht ansatzweise so viel Zeit / Komplexität wie das Erarbeiten der Lösungen erfordert.“). Beide Varianten sind jeweils in den drei Beispielen im Rahmen unserer Auffassung akzeptabel. Beide Versionen sind für den Leser verständlich, auch wenn teils kleinere Fehler im Satz

vorhanden sind.

Zusammenfassend können wir feststellen, dass ParZu die Variante, in der der Zusatz vor das Verb gezogen wird, besser interpretieren kann und die andere Variante, in der der Zusatz am Ende vorkommt, nicht für die Erkennung aller Zusammenhänge benutzen kann. ParZu scheint im Allgemeinen hier sehr empfindlich zu sein. Im Hinblick auf die Verbesserung des Algorithmus könnte man nach solchen Sätzen suchen und versuchen, diese Sätze in die bessere Form für ParZu umzustellen.

Eingabe als richtig bewertet und falsch transformiert

Der kritische Fall, in dem die Eingabe als richtig bewertet wird und die Transformation falsch ist, tritt in 29 Fällen (13,61 %) auf und muss genauer untersucht werden. Dort wird potenziell das Verständnis des Satzes durch die Transformation verschlechtert und nicht abgefangen.

Ein großer Anteil der darin enthaltenen Sätze ist, wie zu erwarten, ein Teil der Sätze, welche wir im Zusammenhang mit einer korrekten Eingabe und einer von unserem Algorithmus fehlerhaften Transformation in Abschnitt 4.2 untersucht haben. Diese Sätze sind vom Benutzer korrekt eingegeben und von ParZu korrekt verarbeitet, aber von uns falsch umgewandelt worden. Insgesamt sind es 16 von den 29 Fällen, welche auf die fehlerhafte Implementierung zurückzuführen ist. Wie diese behoben werden können, wurde schon in Abschnitt 4.2 ausführlich aufgeführt.

In fünf Fällen ist die Benutzereingabe schlicht falsch oder wurde falsch aus D-BAS exportiert. Die Eingaben ergeben keinen vernünftigen Satz (Beispiel: „Ich bin dieser Meinung, weil Kaffeemaschinen aufstellen, wo Studenten sich Kaffee machen können.“) oder beinhalten Export-Fehler (Beispiel: „Ich bin dieser Meinung, weil das & quot nur-Bestehen& quot und der nicht-angepasste Stoff auf die Motivation und den Lernwillen drücken.“). Die „quot“-Ausdrücke werden als Eigennamen identifiziert und ParZu entwickelt daraus zwangsweise falsche und unvollständige Abhängigkeiten.

In drei weiteren Fällen führt unsere Metrik zu einer Abweichung der beiden Bewertungen. Wir haben diese drei Eingaben als falsch bewertet, da einzelne Wörter nicht korrekt angegeben worden sind oder der Satz an sich schwer verständlich ist. ParZu hingegen ist es zum Beispiel egal, ob „MUSS“ in dieser Form angegeben wird (Beispielsatz: „Ich bin dieser Mei-

nung, weil Mit soviel Geld MUSS sich da was machen lassen.“) oder dass die „5 Versuche regel“ getrennt und zum Teil klein geschrieben wird (Beispielsatz: „Ich bin dieser Meinung, weil die 5 Versuche regel für ein neues Fach sowieso neu besprochen werden müsste.“).

In den letzten fünf Fällen ist die Grammatik nicht korrekt. ParZu scheint dennoch mit der falschen Form zurechtzukommen. Anzumerken ist, dass die Erkennung der Abhängigkeiten an diesen Sätzen eher zufällig vervollständigt wird und nicht immer korrekt ist. Beispielsweise wird im Satz „Ich bin dieser Meinung, weil Weil bereits Maschinen da sind und die Anforderungen in den letzten Jahren eigentlich nicht gestiegen sind.“ das zweite „Weil“ als Dativ interpretiert.

Im nächsten Satz „Ich bin dieser Meinung, weil es gibt viele Ansätze und Beweise die für uns Informatiker schlichtweg irrelevant sind.“ ist die Satzstellung falsch und es fehlt ein Komma. Dennoch werden die Abhängigkeiten in diesem Beispiel größtenteils korrekt angegeben und zumindest wird jedem Wort eine Zugehörigkeit zugeordnet. Auffällig ist, dass der dazugehörige grammatikalisch korrekte Satz „Ich bin dieser Meinung, weil es viele Ansätze und Beweise gibt, die für uns Informatiker schlichtweg irrelevant sind.“ dazu führt, dass der letzte Einschub nach dem Verb „gibt“ nicht korrekt erkannt wird. Dieses Problem von ParZu haben wir vorhin schon festgestellt und kann an dieser Stelle nicht alleine durch das Vorziehen des Einschubs behoben werden. Beim zusätzlichen Entfernen des Wortes „uns“, wird der Satz „Ich bin dieser Meinung, weil es viele Ansätze und Beweise, die für Informatiker schlichtweg irrelevant sind, gibt.“ von unserem Filter, wie der originale Satz, als korrekt angesehen.

Zusammenfassend können wir sagen, dass der primitive Ansatz in dieser Form erstaunlich gute Ergebnisse liefert. In 161 von 213 Fällen (75,59 %) stuft unser Filter die Eingabe passend ein. Der relative Anteil an korrekten Transformationen ist mit dem Filter deutlich höher, wenn wir die nicht umgewandelten Sätze nicht mit in die Wertung aufnehmen und liegt bei 80,54 %. Wenn wir nur die Kombination von einer richtigen Eingabe und falscher Transformation als kritisch betrachten, können wir mit dieser Methode die Anzahl an schlechten Umwandlungen auf 29 (13,61 %) senken. Wir haben diese Sätze untersucht und für fast alle Fälle können wir kleinere Verbesserungen am Filter vornehmen, sodass diese auch richtig eingeordnet werden. Allerdings werden auch 23 Sätze (10,8 %) nicht umgewandelt, obwohl der Algorithmus diese korrekt umwandeln kann. Bei diesen Sätzen ist es zum Teil schwierig, Anpassungen vorzunehmen, um alle Fälle korrekt einzuordnen. Dennoch ist auch hier Verbesserungspotential vorhanden.

Wir glauben, dass durch die Komplexität der deutschen Sprache, der Vielfalt von Fehlermöglichkeiten und den vielversprechenden Resultaten, der Ansatz in seiner oder einer ähnlichen Form mit dem Algorithmus verknüpft werden sollte, um eine verwertbare Transformations-Implementierung zu erhalten. Wir denken weiterhin, dass der Kompromiss zwischen dem erhöhten Wert des relativen Anteils von korrekten Transformationen und der Nichtumwandlung von Sätzen eingegangen werden kann und der Nutzen den Nachteil überwiegt. Einen solchen Ansatz weiterzuentwickeln ist keinesfalls eine leichte Aufgabe und bedarf weiterer Forschung.

In Kapitel 4 konnten wir die Ausgabe aus unserem Algorithmus auswerten und genauer erläutern. Wir haben festgestellt, dass ungefähr 67 % der Eingabesätze korrekt umgewandelt werden. Nach der Untersuchung der falschen Transformationen konnten wir die Fehler in lösbar und nicht lösbar beziehungsweise schwer lösbar kategorisieren. Zudem konnten wir die Ursachen der Fehler feststellen. Weiterhin konnten wir die Grenzen des Ansatzes aufzeigen. Zum Vergleich zu der vorherigen Arbeit [Bor20] haben wir eine deutliche Verbesserung von über 50 % erzielt, wobei laut unseren Metriken Borodkins Ansatz ungefähr 40 % und unser Ansatz 67 % korrekt umwandelt. Diese reicht dennoch nicht aus, um den Algorithmus in der Praxis einzusetzen. Es ist deutlich, dass für eine nahezu 100%ig korrekte Umwandlung aller Eingaben viel Arbeit in die Entwicklung eines solchen Algorithmus gesteckt werden muss. Letztlich konnten wir einen vielversprechenden Ansatz zum Filtern der zu transformierenden Benutzereingaben zeigen, um einen höheren Wert des relativen Anteils von korrekten Transformationen zu erzielen.

Kapitel 5

Fazit und Ausblick

In dieser Arbeit haben wir einen Ansatz zur automatischen Umwandlung von Neben- in Hauptsätzen vorgestellt, evaluiert und mit einer bestehenden Arbeit verglichen.

Anfangs haben wir uns mit den Grundlagen zur Entwicklung eines solchen NLP-basierten Ansatzes beschäftigt. Wir haben die Grundlagen wie Testsets und Trainingssets zur Auswertung eines Algorithmus auf diesem Gebiet kennengelernt. Wir konnten zeigen, dass POS-Tagger und Dependency Parser einen Satz in Wortgruppen und Funktionen aufschlüsseln können und haben erklärt, inwiefern wir diese Tools für die Implementierung eines Algorithmus zur automatischen Umwandlung von Neben- in Hauptsätze verwenden können.

Anschließend haben wir anhand des Trainingssets einen Algorithmus zur automatischen Umwandlung von Neben- in Hauptsätze entwickelt. Dabei sind wir auf einige Schwierigkeiten in der Entwicklungsphase eingegangen und konnte zum Abschluss des Algorithmus von den insgesamt 269 Eingaben im Trainingsset nur zwei (für uns korrekte) Sätze nicht umwandeln.

Unsere Evaluation zeigt, dass der Ansatz mithilfe der NLP-Tools POS-Tagger und Dependency Parser vielversprechende Ergebnisse liefert. Im Vergleich zur Arbeit von Borodkin konnten bessere Ergebnisse erzielt werden. Die ungefähre korrekte Transformationsrate von 67 % ist dennoch nicht genug, um den Ansatz in der aktuellen Form für den richtigen Einsatz freizugeben. Wir sind zuversichtlich, dass der Ansatz eine gute Basis darbietet und mit weiteren Methoden deutlich verbessert werden kann. Eine Methode zur Filterung von korrekten und falschen Benutzereingaben haben wir in Abschnitt 4.4 kennengelernt. Wenn wir die bei-

den Methoden verbinden, kommen wir insgesamt auf einen höheren relativen Anteil (80,5 %) von korrekten Transformationen, wenn wir die nicht umgewandelten Sätze nicht mit in die Wertung aufnehmen. Den dort angesprochenen Kompromiss zwischen diesem erhöhten Wert und der Nichtumwandlung von schlechten beziehungsweise nicht erkannten Benutzereingaben können wir im Hinblick auf den hohen Nutzen eingehen. Um ein noch besseres Ergebnis zu erzielen, ist es ratsam, die Schlüsse aus dieser Arbeit in die Implementierung mit aufzunehmen. Zudem kann der primitive Ansatz zur Filterung der Eingaben mit Regeln sinnvoll erweitert werden, sodass dieser noch genauer arbeitet.

Viel Zeit in der Entwicklung eines solchen Algorithmus geht zum Einen in die Implementierung und in das Hinzufügen von Regeln. Der Datensatz muss zuerst auf falsche Umwandlungen überprüft werden. Danach müssen wir eine händische Regel implementieren, welche dafür sorgt, dass auch dieser falsch konvertierter Satz korrekt umgewandelt wird. Anschließend müssen nochmals alle Sätze durchgegangen werden, um zu überprüfen, ob die neue Regel Einfluss auf andere Sätze hat und dazu führt, dass andere Sätze, die vorher korrekt umgewandelt worden sind, jetzt nicht mehr korrekt umgewandelt werden können. Dieser Balanceakt wird zunehmend durch falsche Benutzereingaben und falschen Dependency Parsing erschwert.

Zum Anderen nimmt die Evaluierung einer solchen Implementierung die meiste Zeit in Anspruch. Sowohl die Eingabe der Benutzer als auch die Ausgabe aus dem Dependency Parser und die Ausgabe aus unserem Algorithmus muss untersucht werden. Wir müssen für jeden Fall manuell prüfen, ob dieser Satz korrekt ist und falls der Satz nicht korrekt ist, müssen wir zusätzlich überprüfen, an welcher Stelle des gesamten Vorgangs der Algorithmus scheitert. Zusätzlich ist es wichtig, einzustufen, ob der Fehler (leicht) behoben werden kann. Die zusätzliche Überprüfung unterschiedlicher Metriken, der Ergebnisse aus der Arbeit von Borodkin [Bor20] und unseres Ansatzes zur Filterung von Benutzereingaben vervielfältigt den Aufwand. Dieser enorme Aufwand ist auch der Grund dafür, dass nicht mehr Datensätze getestet wurden. Es ist ratsam, die Größe des Trainingssets und des Testset zu vergrößern, um zum Einen mehr und genauere Regeln für die Transformation erstellen zu können und zum Anderen diese Regeln besser auswerten zu können.

Abgesehen von dem von uns vorgeschlagenen Ansatz zur Filterung der Benutzereingaben gibt es noch viele weitere Möglichkeiten, die Genauigkeit zu erhöhen. Wir können sogar noch vor der Filterung der Benutzereingaben, diese auf Rechtschreibfehler überprüfen und versuchen, diese zu beheben. Zudem könnte eine Datenbank eingebunden werden, welche

Abkürzungen erkennt und diese ausschreibt, anstatt „nur“ den Punkt zu entfernen. Die „Bereinigung“ von Benutzereingaben scheint einen guten Einfluss auf die Transformation zu haben.

Generell können andere Dependency Parser und POS-Tagger getestet werden, um eine höhere Erkennungsrate von Sätzen zu erhalten. Wir haben im Kapitel 1 einige POS-Tagger und Dependency Parser kennengelernt, welche ebenfalls gut für diese Aufgabe geeignet sein könnten. Allerdings muss bei einer neuen Wahl solcher Tools der Algorithmus entsprechend auf die Eigenschaften und Ungenauigkeiten dieser grundlegend neu abgestimmt werden.

Dieser Ansatz kann zudem auch zur Umwandlung von Haupt- in Nebensätze genutzt werden. Die Regeln müssen zwar dementsprechend angepasst werden, aber die generelle Vorgehensweise bleibt dieselbe. Die Struktur der Sätze muss erkannt, das Verb gefunden und an die richtige Stelle platziert werden.

Das Thema als solches ist keineswegs vollständig erforscht und auch nicht komplett ausgeschöpft. Dieser Ansatz bietet verschiedene Anwendungsmöglichkeiten. Zum Beispiel kann der Ansatz für D-BAS-basierte Anwendungen, Anwendungen zur Umwandlung von Texten in eine leichte Sprache oder Anwendungen, welche Argumente herausfiltern und speichern, benutzt werden. Wir halten weitere Forschungen zu diesem Thema für wichtig, angemessen und vielversprechend.

Literatur

[Bor20] Kirill Borodkin. *Entwicklung einer Bibliothek zur automatisierten Umformulierung von Neben- in Hauptsätze*. Bachelor's thesis. Aug. 2020.

[Fot06] Kilian A Foth. „Eine umfassende Constraint-Dependenz-Grammatik des Deutschen“. 2006.

[Hon+16] Matthew Honnibal u. a. *spacy: Industrial-strength natural language processing in python*. Eingesehen am 05.03.2021. 2016. URL: https://spacy.io/.

[Kar+18] Elif Kara u. a. „A domain-adapted dependency parser for german clinical text“. In: *Proceedings of the 14th Conference on Natural Language Processing (KONVENS)*. Verlag der Österreichischen Akademie der Wissenschaften, 2018, S. 12–17.

[KMM17] Tobias Krauthoff, Christian Meter und Martin Mauve. „Dialog-Based Online Argumentation: Findings from a Field Experiment“. In: *Proceedings of the 1st Workshop on Advances in Argumentation in Artificial Intelligence*. Bari, Nov. 2017, S. 85–99.

[Kra+18] Tobias Krauthoff u. a. „D-BAS – A Dialog-Based Online Argumentation System“. In: *Computational Models of Argument*. Warsaw, Sep. 2018, S. 325–336.

[Man+14] Christopher D Manning u. a. „The Stanford CoreNLP natural language processing toolkit“. In: *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*. 2014, S. 55–60.

[McD+05] Ryan McDonald u. a. „Non-projective dependency parsing using spanning tree algorithms“. In: *Proceedings of human language technology conference and conference on empirical methods in natural language processing*. 2005, S. 523–530.

[Men] *Text automatisch online umschreiben*. Eingesehen am 29.04.2021. URL: https://www.text-online-umschreiben.de/.

[NHN06] Joakim Nivre, Johan Hall und Jens Nilsson. „MaltParser: A Data-Driven Parser-Generator for Dependency Parsing“. In: *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*. Genoa, Italy: European Language Resources Association (ELRA), Mai 2006.

[Niv+07] Joakim Nivre u. a. „The CoNLL 2007 shared task on dependency parsing“. In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. 2007, S. 915–932.

[Ope11] Apache OpenNLP. *Apache software foundation*. Eingesehen am 28.04.2021. 2011. URL: `http://opennlp.apache.org`.

[ORD19] Katrin Ortmann, Adam Roussel und Stefanie Dipper. „Evaluating Off-the-Shelf NLP Tools for German.“ In: *Proceedings of the 15th Conference on Natural Language Processing (KONVENS)*. 2019.

[Pre] *Prepostseo: Text umschreiben*. Eingesehen am 29.04.2021. URL: `https://www.prepostseo.com/de/paraphrasing-tool`.

[Pro18] Thomas Proisl. „SoMeWeTa: A part-of-speech tagger for German social media and web texts“. In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC)*. 2018.

[Sch13] Helmut Schmid. „Probabilistic part-of-speech tagging using decision trees“. In: *New methods in language processing*. 2013, S. 154.

[Sen+09] Rico Sennrich u. a. „A new hybrid dependency parser for German“. In: *Proceedings of the German Society for Computational Linguistics and Language Technology* (2009), S. 115–124.

[SL08] Helmut Schmid und Florian Laws. „Estimation of conditional probabilities with decision trees and an application to fine-grained POS tagging“. In: *Proceedings of the 22nd International Conference on Computational Linguistics (Coling)*. 2008, S. 777–784.

[Smo] *Smodin: Volltext automatisch umschreiben*. Eingesehen am 29.04.2021. URL: `https://smodin.me/de/Text-automatisch-in-deutscher-Sprache-umformulieren`.

[SVS13] Rico Sennrich, Martin Volk und Gerold Schneider. „Exploiting synergies between open resources for german dependency parsing, pos-tagging, and morphological analysis“. In: *Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP*. 2013, S. 601–609.

[UHH16] Erik Faessler Udo Hahn Franz Matthies und Johannes Hellrich. „UIMA-Based JCoRe 2.0 Goes GitHub and Maven Central — State-of-the-Art Software Resource Engineering and Distribution of NLP Pipelines“. In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation*. 2016, S. 1–8.

[Zem+18] Daniel Zeman u. a. „CoNLL 2018 shared task: Multilingual parsing from raw text to universal dependencies“. In: *Proceedings of the CoNLL 2018 Shared Task: Multilingual parsing from raw text to universal dependencies*. 2018, S. 1–21.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, 06. Mai 2021 Sebastian Bischoff