

FERNUNIVERSITÄT HAGEN

BACHELOR-ARBEIT

---

# Automatische Versmaßannotation in Werken der homerischen Hexameter-Dichtung

---

*Autorin:*

Dr. Anne-Kathrin Schumann

*Betreuer:*

Prof. Dr. Christoph Beierle  
Prof. Dr. Norbert Blößner

*Abschlussarbeit  
zur Erlangung des Grades B. Sc. (Informatik)*

*am*

Lehrgebiet Wissensbasierte Systeme  
Fakultät für Mathematik und Informatik

*in Kooperation mit dem*

Institut für Griechische und Lateinische Philologie  
der Freien Universität Berlin

3. Juni 2020  
(leicht bearbeitete Fassung)





# Inhaltsverzeichnis

<b>Zusammenfassung</b>	<b>v</b>
<b>1 Einleitung</b>	<b>1</b>
<b>2 Die griechische Sprache und das Problem der Versmaßannotation</b>	<b>3</b>
2.1 Vorbemerkung	3
2.2 Grundlegendes zur griechischen Sprache der homerischen Epoche	3
2.3 Der homerische Hexameter	4
2.3.1 Der Begriff des Hexameters	4
2.3.2 Besonderheiten der homerischen Hexameter-Dichtung	5
2.3.3 Regeln für die Versmaßannotation	7
2.4 Ausgangsdaten	9
<b>3 Forschungsstand</b>	<b>11</b>
3.1 Vorbemerkung	11
3.2 Allgemeines zur Methodik und zur Evaluation	11
3.3 Automatische Annotation des Wortakzents	14
3.4 Untersuchung klanglicher Eigenschaften lyrischer Werke	15
3.5 Automatische Versmaßannotation	16
3.6 Implementierungen	21
3.7 Diskussion und Auswahl eines Implementierungsansatzes	23
<b>4 Automatische Versmaßannotation</b>	<b>25</b>
4.1 Vorbemerkung	25
4.2 Nutzung endlicher Automaten für computerlinguistische Fragestellungen	25
4.3 Definitionen	26
4.3.1 Deterministischer endlicher Automat	26
4.3.2 Finite-State-Transducer	27
4.3.2.1 Definition	27
4.3.2.2 Operationen und Anwendung	28
4.4 Annotationsprogramm	29
4.4.1 Allgemeiner Annotationsalgorithmus	29
4.4.2 Pakete und Klassen	33
4.4.2.1 Pakete	33
4.4.2.2 Klassengeflecht	34
4.4.2.3 Klasse <i>preprocessor</i>	35
4.4.2.4 Klasse <i>ruleset</i>	36
4.4.2.5 Klasse <i>verse</i>	37
4.4.2.6 Klasse <i>annotator</i>	37
4.4.2.7 Hierarchische endliche Automaten	39
4.4.2.8 Klasse <i>transducer</i>	42
4.5 Zusammenfassung	45

<b>5</b>	<b>Evaluation</b>	<b>47</b>
5.1	Vorbemerkung . . . . .	47
5.2	Evaluationsdaten, Metriken und Baseline . . . . .	47
5.3	Silbentrennung . . . . .	48
5.4	Versmaß-Annotation . . . . .	49
5.5	Diskussion . . . . .	49
<b>6</b>	<b>Anwendung</b>	<b>51</b>
6.1	Vorbemerkung . . . . .	51
6.2	Annotation des Gesamtkorpus . . . . .	51
6.3	Qualität der Gesamtannotation mit WinMetrix . . . . .	52
6.4	Übereinstimmungen . . . . .	53
6.5	Kennzahlen für die eigene Annotation des Gesamtkorpus . . . . .	53
6.6	Diskussion . . . . .	55
<b>7</b>	<b>Diskussion und Ausblick</b>	<b>57</b>
	<b>Abbildungsverzeichnis</b>	<b>59</b>
	<b>Tabellenverzeichnis</b>	<b>61</b>
	<b>Definitionsverzeichnis</b>	<b>63</b>
	<b>Glossar</b>	<b>65</b>
	<b>Literatur</b>	<b>69</b>

## **Zusammenfassung**

### **Automatische Versmaßannotation in Werken der homerischen Hexameter-Dichtung**

von Dr. Anne-Kathrin Schumann

Die vorliegende Arbeit widmet sich der Entwicklung, Implementierung und Anwendung eines Verfahrens zur automatisierten, computergestützten Versmaßannotation in Texten der frühgriechischen Epik. Zu diesem Zweck führt die Arbeit zunächst in die Themenstellung ein, diskutiert deren Relevanz und arbeitet die im Einzelnen zu untersuchenden Teilprobleme heraus. Alsdann folgt eine kurze Einführung in die griechische Sprache, wobei sich die Darstellung jedoch auf die Dokumentation derjenigen linguistischen Regeln, welche die Grundlage für die später zu leistende Implementierung bilden sollen, konzentriert. Des Weiteren werden die Eigenschaften der zu annotierenden Ausgangsdaten diskutiert. Im Anschluss daran stellt die Arbeit aktuelle Lösungsansätze aus der computerlinguistischen Forschung vor, von denen schließlich ein auf endlichen Automaten beruhender und damit in der Computerlinguistik gut etablierter und vielfältig eingesetzter Ansatz für den Entwurf eines Annotationsalgorithmus herangezogen wird. Die auf diese Weise erzielten Annotationsergebnisse werden in der Folge im Hinblick auf Annotationsgenauigkeit und -abdeckung evaluiert und diskutiert, so dass der Algorithmus schließlich für die Annotation eines kompletten Textkorpus genutzt werden kann. Die Arbeit setzt einen informatisch vorgebildeten Leser voraus, trägt dem interdisziplinären Charakter des behandelten Themas jedoch in Form eines Glossars Rechnung, das Definitionen der verwendeten linguistischen, computerlinguistischen und literaturwissenschaftlichen Fachbegriffe enthält. Der im Rahmen der Arbeit erstellte Python-Code ist als Open-Source-Projekt unter der URL [https://github.com/anetschka/greek\\_scansion](https://github.com/anetschka/greek_scansion) abrufbar.

Am Abendhimmel blühet ein Frühling auf;  
 Unzählig blühn die Rosen und ruhig scheint  
 Die goldne Welt; o dorthin nimmt mich,  
 Purpurne Wolken! und möge droben

In Licht und Luft zerrinnen mir Lieb und Leid! –  
 Doch, wie verscheucht von töriger Bitte, flieht  
 Der Zauber; dunkel wirds und einsam  
 Unter dem Himmel, wie immer, bin ich –

Friedrich Hölderlin, aus dem Gedicht „Abendphantasie“<sup>1</sup>

Friedrich Hölderlin hat wunderschöne Gedichte geschrieben. Da ich das Griechische nicht beherrsche und zur Lösung der gestellten Aufgabe auf die Hilfe von Experten angewiesen war, hatte die Hölderlinsche Lyrik für mich im Kontext dieser Arbeit auch die Funktion, zumindest im Ansatz zu verstehen, welche Effekte Metrik auf die Wirkung und Ausdruckskraft von Gedichten haben kann. Es scheint mir daher folgerichtig, an den geeigneten Stellen statt ausdrucksarmer Floskeln Zitate des wortgewaltigeren Dichters sagen zu lassen, was neben dem eigentlichen Inhalt des Textes noch von Bedeutung sein mag. Sofort und ausdrücklich möchte ich mich indes an dieser Stelle bei Jonas Sültmann und Simon Neumaier bedanken, die mir unzählige Fragen zur griechischen Sprache beantwortet und meine Evaluationsdaten annotiert haben. Ohne ihre Unterstützung hätte ich diese Arbeit nicht schreiben können.

Ich widme die Arbeit meiner Familie.

---

<sup>1</sup>Die Hölderlin-Zitate in dieser Arbeit sind der Gedichtsammlung Hölderlin (1977) entnommen.

## Kapitel 1

# Einleitung

Die vorliegende Arbeit befasst sich mit der automatischen Annotation des Versmaßes in den erhaltenen Texten der frühgriechischen Epik. Diese umfassen die Homer zugeschriebenen Epen Odysee und Ilias, die Texte Hesiods und die frühgriechischen Hymnen. Inhaltlich und methodologisch lässt sich die behandelte Themenstellung dem interdisziplinären Forschungsbereich der digitalen Geisteswissenschaft (*Digital Humanities*) zuordnen, für den die folgenden Ziele zentral sind:

- die Digitalisierung geisteswissenschaftlicher Forschungsdaten, d. h. die Umwandlung der Daten in maschinenlesbare Formate,
- die Auswertung maschinenlesbarer geisteswissenschaftlicher Forschungsdaten mit Hilfe computergestützter Methoden,
- die Publikation geisteswissenschaftlicher Forschungsdaten und Forschungsergebnisse mit Hilfe moderner Computertechnik mit dem Zweck, Forschung transparent und reproduzierbar zu machen und Arbeitsergebnisse einem möglichst breiten Kreis potenzieller Nutzer zu präsentieren, und
- die Befähigung von Geisteswissenschaftlern zur Nutzung computergestützter Methoden durch Aus- und Weiterbildung.

Die Hinwendung geisteswissenschaftlicher Disziplinen zu einem häufig nicht nur „digitalen“, sondern auch datengetriebenen Forschungsparadigma ist dabei durchaus ambivalent: Den offensichtlichen Vorteilen beispielsweise in der Aufbereitung und Publikation zentraler Ressourcen und Forschungsinhalte stehen die mit mathematisch motivierten Algorithmen verbundenen Unwägbarkeiten gegenüber. In jedem Fall ist mit der Nutzung moderner Computertechnik und der damit einhergehenden Adaption datenorientierter Untersuchungsmethoden ein nicht zu unterschätzendes disruptives Potential verbunden (vgl. etw. die ausführliche Diskussion in Larsen-Freeman und Cameron (2008) zur Entwicklung der Linguistik).

Der vorliegenden Arbeit allein ein disruptives oder gar „revolutionäres“ Potential zuzuschreiben wäre indes abwegig. Das untersuchte Problem – die Ergänzung eines bereits vorhandenen Korpus<sup>1</sup> (Kruse, 2014, vgl. hierzu auch Abschnitt 2.4) homerischer Texte um eine Annotation des Versmaßes für jeden einzelnen Vers – gehört vielmehr zu den Grundlagenproblemen der „digitalen“ Altphilologie und kann damit am ehesten den Punkten eins und drei der obigen Aufzählung zugeschlagen werden. Außerdem wird in dieser Arbeit das Ziel verfolgt, bereits vorhandenes altphilologisches Regelwissen explizit in die algorithmische Lösung einzubeziehen,

---

<sup>1</sup>Der Begriff des „Korpus“ selbst enthält keine Information darüber, in welcher Form das Korpus existiert. Dennoch wird in dieser Arbeit, soweit nicht explizit anderweitig qualifiziert, ein Korpus immer als maschinenlesbare Textmenge aufgefasst, vgl. auch den entsprechenden Glossareintrag.

d. h. die resultierende Software wird transparente, da an fachlich motivierte Regeln geknüpfte und damit für einen menschlichen Nutzer nachvollziehbare Annotationsentscheidungen treffen. Dafür ist eine Form der Wissensrepräsentation zu entwickeln, die als Grundlage für den zu implementierenden Annotationsalgorithmus dienen kann, dabei aber für den altphilologisch geschulten Experten verständlich bleibt. Vorhandenes Expertenwissen ist mithin geeignet zu dokumentieren, zu formalisieren und zu implementieren. Im Einzelnen lässt sich die Problemstellung in die folgenden Teilprobleme aufgliedern:

1. Die vollständige Beschreibung des vorhandenen altphilologischen Expertenwissens. Dies wird in Abschnitt 2.3.3 geleistet.
2. Die Wahl einer geeigneten Methode zur Formalisierung des unter 1. beschriebenen Regelwissens aus Sicht der Wissensrepräsentation. Das Ziel dieser Formalisierung ist die Entwicklung einer Repräsentationsform für das altphilologische Expertenwissen, die sowohl als Grundlage für einen Annotationsalgorithmus als auch zur Information eines menschlichen Spezialisten dienen kann. Die gewählte Formalisierung mittels endlicher Automaten wird sowohl aus theoretischer Sicht als auch im Hinblick auf die implementierte Annotations-Software in Kapitel 4 ausführlich beschrieben.
3. Die Implementierung eines Annotationsalgorithmus, der in der Lage ist, das Versmaß eines altgriechischen Hexameter-Verses zu bestimmen. Der Algorithmus soll die unter 2. entwickelte Form der Wissensrepräsentation nutzen. Erfahrungswerte besagen, dass sich 90 % (Höflmeier, 1982, S. 7) aller Verse eindeutig hinsichtlich ihres Hexameter-Typs bestimmen lassen. Die Annotations-Software wird in Abschnitt 4.4 beschrieben.
4. Die Anwendung des unter 3. entwickelten Algorithmus auf das von Kruse (2014) aufbereitete Korpus homerischer Hexameter-Epen zum Zweck der Evaluation des Algorithmus und der Erweiterung der vorhandenen Datenbasis. Nicht (eindeutig) annotierbare Verse sollen dabei dokumentiert werden. Ablauf und Ergebnis der Evaluation werden in Abschnitt 5 diskutiert. Das Ergebnis der Gesamtannotation wird dagegen in Abschnitt 6 vorgestellt und diskutiert.

Technologisch strebt die vorliegende Arbeit an, das Problem der zeitaufwendigen manuellen Versmaßannotation einer neuartigen, automatischen Lösung mit sehr hoher Annotationsqualität zuzuführen. Aus Sicht der *Digital Humanities* besteht der Beitrag der Arbeit darin, dass die bereits von Kruse (ebd.) begonnene „Restauration“ veralteter und nicht mehr handelbarer Daten und IT-Artefakte fortgesetzt wird; zudem wird der existierende Datensatz erweitert und damit neuen Forschungsfragen zugänglich gemacht, ein grundlegendes Verfahren des Fachs wird dokumentiert, automatisiert und auf diese Weise der wissenschaftlichen Reproduktion und Evaluation geöffnet. Nicht zuletzt werden für die formalisierte, maschinenlesbare Beschreibung des für die Versmaßannotation erforderlichen Expertenwissens (sehr einfache) Techniken der Wissensrepräsentation genutzt.



## Kapitel 2

# Die griechische Sprache und das Problem der Versmaßannotation

Blüht Ionien? ists die Zeit? denn immer im Frühling,  
 Wenn den Lebenden sich das Herz erneut und die erste  
 Liebe den Menschen erwacht und goldner Zeiten  
 Erinnerung,  
 Komm ich zu dir und grüß in deiner Stille dich, Alter!

Friedrich Hölderlin, aus dem Gedicht „Der Archipelagus“

## 2.1 Vorbemerkung

In diesem Kapitel sollen die Voraussetzungen für die umzusetzende Annotationsaufgabe dargestellt werden, und zwar einerseits aus sprachlich-literarischer Sicht und andererseits aus Sicht der Daten und IT-Artefakte. Zu diesem Zweck beschreibt Abschnitt 2.2 einige grundlegende Eigenschaften der griechischen Sprache zur Entstehungszeit der frühgriechischen Epen. Abschnitt 2.3 definiert den Begriff des „Hexameters“, geht auf die Besonderheiten des homerischen Hexameters ein und fasst die für die Annotation benötigten Annotationsregeln zusammen. Abschnitt 2.4 beschreibt das von Kruse (2014) erstellte Korpus homerischer Vers-Epen, das die Datengrundlage für die hier durchgeführte Untersuchung bildet.

## 2.2 Grundlegendes zur griechischen Sprache der homerischen Epoche

Die griechische Sprache weist eine im Vergleich zu anderen klassischen Sprachen bemerkenswerte Kontinuität im schriftlichen und mündlichen Gebrauch auf und hat im Verlauf ihrer mehrere Jahrtausende umfassenden Geschichte unterschiedliche Entwicklungsstadien durchlaufen. Die in vorliegender Arbeit untersuchten Texte lassen sich im Hinblick auf ihre Entstehungszeit dem von Bußmann (1990) als „Klassisches Griechisch“ bezeichneten Entwicklungsstadium zuordnen, das zeitlich zwischen einer älteren, archaischen Entwicklungsstufe und dem hellenistischen Griechisch des alexandrinischen Reiches einzuordnen ist. Als systemische Besonderheiten dieser mittleren Entwicklungsstufe gelten die folgenden Merkmale (ebd.):

- ein komplexes Vokalsystem mit langen und kurzen Vokalen und Diphthongen,
- musikalischer Wortakzent mit semantisch distinktiver Stimmtönenbewegung auf einer Wortsilbe,

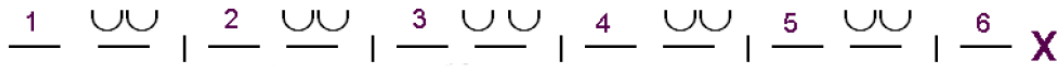


ABBILDUNG 2.1: Allgemeines Annotationsschema für Hexameter-Verse.

- komplexes Kasussystem mit 5 Kasus,
- komplexes Numerussystem mit Auftreten des Dualis.

Wesentlich für die hier zu bewältigende Annotationsaufgabe sind insbesondere die prosodischen Eigenschaften des Griechischen. So bewirkt der musikalische Wortakzent, dass die Betonung der Silben nicht wie im Deutschen durch Verstärkung des Tones bewirkt wird, sondern durch Aussprache in höherer Tonlage (Leggewie u. a., 1981, S. 10): „Auf den musikalischen Akzent weist schon der Antike Ausdruck  $\pi\rho\omicron\sigma\omega\delta\acute{\iota}\alpha$ , lat. *accentus* (cantare) = ‚das Dazu-Singen‘ hin“. Sowohl für die Silbentrennung als auch für die Versmaßannotation sind zudem kontextuelle Markierungen an den Vokalen zu beachten, insbesondere:

- Akzentzeichen, vor allem der Zirkumflex (z. B. über dem Diphthong  $\omicron\upsilon$  im Wort  $\nu\omicron\upsilon\zeta$ ), der lange Vokale oder Diphthonge kennzeichnet (ebd., S. 11),
- Lesezeichen, insbesondere das Trema, das anzeigt, dass der markierte Vokal nicht wie gewöhnlich mit einem davorstehenden Vokal zu einem Diphthong verschmilzt, sondern getrennt ausgesprochen wird und demnach auch eine eigene Silbe bildet (ebd., S. 13, ein Beispiel findet sich im ersten Vers der Ilias:  $\Pi\eta\lambda\eta\grave{\iota}\acute{\alpha}\delta\epsilon\omega$ , getrennt  $\pi\eta.\lambda\eta.\iota.\alpha.\delta\epsilon\omega$ ).

Aus computerlinguistischer Sicht können an dieser Stelle die folgenden Aspekte hervorgehoben werden:

1. Das Griechische weist eine Reihe komplexer linguistischer Eigenschaften auf, welche die automatische Verarbeitung erschweren können (z. B. eine formenreiche Morphologie).
2. Aufgrund seiner langen Entwicklungsgeschichte hat das Griechische unterschiedliche Entwicklungsstadien mit z. T. deutlichen Veränderungen des linguistischen Systems durchlaufen (ebd.). In den Texten Homers können linguistische Merkmale unterschiedlicher Entwicklungsstadien auftreten (Hackstein, 2011a, vgl. hierzu auch Abschnitt 2.3.2) – ein Umstand, dem in der automatischen Verarbeitung möglicherweise Rechnung zu tragen ist.
3. Sowohl bei der Silbentrennung als auch bei der Versmaßannotation selbst sind lokale Kontextinformationen in Form von Diakritika (Zirkumflex und Trema) zu beachten.

## 2.3 Der homerische Hexameter

### 2.3.1 Der Begriff des Hexameters

Der Hexameter (von griech.  $\acute{\epsilon}\xi\acute{\alpha}\mu\epsilon\tau\rho\omicron\nu$  (translit. *hexámetron*) mit *hex* = „sechs“ und *metron* = „Maß“) ist das Versmaß der griechischen und römischen Epik (Leggewie u. a., 1981). Wichtig für das Verständnis des Hexameters ist, dass der Rhythmus nicht

wie im Deutschen durch den Wechsel betonter und unbetonter Silben (dynamisch-akzentuierender Rhythmus) erzeugt wird, sondern durch die „geregelte Abfolge langer und kurzer Silben (quantitierender Rhythmus)“ (ebd.<sup>1</sup>). Hierbei spielen die bereits im vorigen Abschnitt genannten Besonderheiten des Vokalsystems und die „singende“ Prosodie des Griechischen eine wesentliche Rolle.

Der Hexameter besteht aus sechs Versfüßen. Die erste Silbe eines jeden Versfußes ist dabei immer lang (sie bildet ein *longum*). Ihr folgen entweder zwei kurze Silben – in diesem Fall ist der Versfuß als Daktylus realisiert. Folgt der ersten Silbe dagegen eine weitere lange Silbe, bildet der Versfuß einen Spondeus. Hexameter-Verse können, bezogen auf die ersten fünf Versfüße, Daktylen und Spondeen in freiem Wechsel enthalten, der sechste Versfuß besteht dagegen stets aus zwei Silben, wobei die letzte Silbe entweder lang oder kurz ist. Allerdings gibt es im Hinblick auf die Verteilung von Spondeen und Daktylen im Vers signifikante Vorlieben (so werden z. B. Spondeen im fünften Fuß vermieden). Genauere Angaben hierzu macht Papakitsos (2011, S. 61 f.).

Eine graphische Annotation von Hexameter-Verse lässt sich umsetzen, indem lange Silben durch einen Querstrich, kurze Silben durch nach oben offene Bögen und die letzte Silbe des Verses durch ein X (die sogenannte *anceps*) markiert werden. Wenn dabei Annotationsvarianten vertikal gestaffelt dargestellt werden, lässt sich ein allgemeines Annotationsschema für Hexameter-Verse wie in Abbildung 2.1 angeben<sup>2</sup>. Insgesamt ergeben sich aufgrund der Variabilität der ersten fünf Versfüße  $2^5 = 32$  annotierbare Hexameter-Varianten (vgl. Abb. 2.2). Die in der Abbildung den einzelnen Varianten zugeordneten Nummern bezeichnen in ihrer ersten Ziffer die Anzahl der Spondeen im Vers, die zweite Ziffer ist dagegen eine laufende Nummer (Höflmeier, 1982).

### 2.3.2 Besonderheiten der homerischen Hexameter-Dichtung

Der langen Rezeptions- und Wirkungsgeschichte der homerischen Epen entspricht eine aus wissenschaftshistorischer Sicht ebenfalls lange Geschichte der Erforschung der Texte Homers. Ein umfassender Überblick über Forschungsarbeiten zur sprachlichen Gestalt der Odyssee und der Ilias ist daher schon aufgrund der schier unendlichen Menge an relevanten Arbeiten hier nicht zu leisten. Zumindest sollen jedoch an dieser Stelle diejenigen Aspekte, die sich als für die Versmaßannotation relevant erweisen können, kurz benannt und erläutert werden. Für weiterführende Informationen sei auf Überblicksdarstellungen wie die Arbeit von Rengakos und Zimmermann (2011) verwiesen, die Einstiegspunkte in die vertiefte Beschäftigung mit allen hier behandelten Aspekten bietet.

Als erste Besonderheit ist das Changieren der Texte Homers zwischen den linguistischen Polen Mündlichkeit/Schriftlichkeit hervorzuheben. Sowohl die Ilias als auch die Odyssee sind heutigen Lesern selbstverständlich als schriftlich fixierte Texte bekannt. Gleichwohl verweisen sprachliche Merkmale beider Texte deutlich auf einen mündlichen Darbietungs- und Überlieferungskontext (Rösler, 2011):

<sup>1</sup>Leggewie u. a. (1981, S. 258) verweisen darauf, dass „... moderne Menschen des europäischen Kulturkreises im allgemeinen nicht in der Lage sind, einen Versrhythmus durch quantitierendes Lesen zum Ausdruck zu bringen“.

<sup>2</sup>Die vertikalen Striche in der Abbildung sind optional und sollen lediglich die sechs Versfüße deutlicher von einander abgrenzen. Hervorzuheben ist in diesem Zusammenhang jedoch, dass Versfuß- und Wortgrenzen nicht übereinstimmen müssen, d. h. ein Wort kann sich über mehrere Versfüße erstrecken.

—vv —vv —vv —vv —vv —X	00
—vv —vv —vv —vv —vv —X	10
—vv —vv —vv —vv —vv —X	11
—vv —vv —vv —vv —vv —X	12
—vv —vv —vv —vv —vv —X	13
—vv —vv —vv —vv —vv —X	14
—vv —vv —vv —vv —vv —X	20
—vv —vv —vv —vv —vv —X	21
—vv —vv —vv —vv —vv —X	22
—vv —vv —vv —vv —vv —X	23
—vv —vv —vv —vv —vv —X	24
—vv —vv —vv —vv —vv —X	25
—vv —vv —vv —vv —vv —X	26
—vv —vv —vv —vv —vv —X	27
—vv —vv —vv —vv —vv —X	28
—vv —vv —vv —vv —vv —X	29
—vv —vv —vv —vv —vv —X	30
—vv —vv —vv —vv —vv —X	31
—vv —vv —vv —vv —vv —X	32
—vv —vv —vv —vv —vv —X	33
—vv —vv —vv —vv —vv —X	34
—vv —vv —vv —vv —vv —X	35
—vv —vv —vv —vv —vv —X	36
—vv —vv —vv —vv —vv —X	37
—vv —vv —vv —vv —vv —X	38
—vv —vv —vv —vv —vv —X	39
—vv —vv —vv —vv —vv —X	40
—vv —vv —vv —vv —vv —X	41
—vv —vv —vv —vv —vv —X	42
—vv —vv —vv —vv —vv —X	43
—vv —vv —vv —vv —vv —X	44
—vv —vv —vv —vv —vv —X	50

ABBILDUNG 2.2: Systematischer Überblick über Hexameter-Varianten (Höflmeier, 1982, S. 10).

„Die Formelhaftigkeit der homerischen Sprache und bestimmte Strukturelemente des epischen Erzählens ... lassen sich nur unter der Voraussetzung verstehen, dass derartige Dichtung ursprünglich improvisierend hervorgebracht wurde ....“

Für die Altphilologie hat diese Feststellung ungeachtet ihres möglicherweise wenig überraschenden Charakters weitreichende Implikationen, beispielsweise für die Datierung der homerischen Epen und die Erforschung des kulturellen Status epischer Dichtung im antiken Griechenland (vgl. ebd.). Aus der Sicht des in dieser Arbeit untersuchten Problems der Versmaßannotation ist hervorzuheben, dass gerade die Merkmale der epischen Sprache sowohl zur Konservierung linguistischer Archaismen als auch zur Entwicklung von Neuerungen, d. h. zu einem Nebeneinander sprachlicher Formen führen, die ganz unterschiedlichen Entwicklungsepochen der griechischen Sprache angehören (Hackstein, 2011a):

„Die homerische Sprachform ist janus-köpfig. Metrik und Formelsprache konservieren ältere Formen, fördern aber gleichzeitig die Bildung jüngerer Formen. Einerseits bewahrt die epische Sprache erstaunliche Altertümlichkeiten, andererseits besitzt sie die Fähigkeit zu Neuerungen, die selbst im späteren Attischen ohne Gegenstück bleiben ....“

So führt Hackstein (ebd., vgl. auch Hackstein, 2011b) Beispiele für morphologische, syntaktische und phonologisch-metrische Archaismen und Varianten an. Darüber hinaus ist zu beobachten, dass aus metrischen Gründen (*metri causa*) Regeln zur Gestaltung des Hexameters bisweilen verletzt werden, also z. B. eigentlich obligatorische Kürzungen unterbleiben. Hackstein (ebd.) bietet hierzu eine ausführliche Diskussion, eine allgemeinverständliche und auch im Hinblick auf aktuell kontrovers diskutierte Forschungsfragen alternative Darstellung sprachlicher Aspekte findet sich in Mannsperger und Mannsperger (2017). Auf ein detailliertes Eingehen auf

Diphthong	Phonetische Transkription
αι	[ai̯]
οι	[oi̯]
υι	[yi̯]
ει	[ei̯]
αυ	[au̯]
ευ	[eu̯]
ου	[ou̯]
ηι	[ɛi̯]
ωι	[ɔi̯]
ηυ	[ɛu̯]

TABELLE 2.1: Griechische Diphthonge nach Papakitsos (2011).

diese Gesichtspunkte soll hier allerdings verzichtet werden. Stattdessen werden die im folgenden Abschnitt angeführten Annotationsregeln als Grundlage für die weitere Arbeit genutzt. Als Zwischenstand kann jedoch festgehalten werden, dass die in Abschnitt 2.3.1 herausgearbeitete Regelmäßigkeit des Hexameters insofern eine gewisse Eintrübung erfährt, als mit dem Auftreten zahlreicher Varianten zu rechnen ist, die gerade dem epischen Charakter der zu verarbeitenden Texte geschuldet zu sein scheinen.

### 2.3.3 Regeln für die Versmaßannotation

Das Grundgerüst der in diesem Abschnitt dargestellten Regeln entstammt der persönlichen Kommunikation mit dem Zweitbetreuer dieser Arbeit, Professor Norbert Blößner und Mitarbeitern, sowie einem bereits existierenden Pascal-Programm, das in der Vergangenheit für die regelbasierte Versmaßannotation genutzt wurde. Die diesem Programm zugrunde liegenden Überlegungen und Regeln sind in Höflmeier (1982) dokumentiert<sup>3</sup>. Der so erarbeitete Grundbestand an Regeln wurde zudem durch Regeln ergänzt, die der sehr detaillierten Darstellung von Papakitsos (2011) entnommen werden konnten. Weitere Hinweise entstammen Leggewie u. a. (1981). Die linguistischen Regeln für die Versmaßannotation befassen sich im Wesentlichen mit der durch Vokale und Diphthonge bestimmten Länge einzelner Silben. Hier wird zwischen „natürlicher“ und positionsbedingter Länge unterschieden. Die Regeln lauten wie folgt:

1. **Regel  $R_{nl1}$  (Naturlänge 1):** Stets lang ist eine Silbe, die den naturlangen Vokal  $\eta$  bzw.  $\eta$  (*eta*, Aussprache in phonetischer Transkription: [ɛi̯]<sup>4</sup>) oder den ebenfalls naturlangen Vokal  $\Omega$  bzw.  $\omega$  (*omega*, [ɔi̯]) enthält. **Ausnahmen** von der Anwendung der Naturlänge sind möglich bei Auftreten eines Hiats (s. u.).
2. **Regel  $R_{nl2}$  (Naturlänge 2):** Stets lang sind auch Silben, die auf einen Diphthong enden (wobei der zweite Vokal auch zum folgenden Wort gehören kann). Tabelle 2.1 gibt einen Überblick über die in Betracht

<sup>3</sup>Über die reine Versmaßannotation hinaus leistet Höflmeier zudem eine statistische Analyse von Versmaßverteilungen auf Grundlage der aus der Anwendung seines Programms resultierenden Annotationen.

<sup>4</sup>Allgemeine Angaben zur griechischen Lautlehre finden sich auch in Leggewie u. a. (1981, S. 5-23.). Die phonetischen Transkriptionen in dieser Arbeit orientieren sich am phonetischen Alphabet der International Phonetic Association (<https://www.internationalphoneticassociation.org/>).

Konsonant	Name	Phonetische Transkription
B, β	beta	[b]
Γ, γ	gamma	[g]
Δ, δ	delta	[d]
Z, ζ	zeta	[dz]
Θ, θ	theta	[t <sup>h</sup> ]
K, κ	kappa	[k]
Λ, λ	lambda	[l]
M, μ	my	[m]
N, ν	ny	[n]
Ξ, ξ	xi	[ks]
Π, π	pi	[p]
P, ρ	rho	[r], [r <sup>h</sup> ]
Σ, σ	sigma	[s], [z]
T, τ	tau	[t]
Φ, φ	phi	[p <sup>h</sup> ]
X, χ	chi	[k <sup>h</sup> ]
Ψ, ψ	psi	[ps]

TABELLE 2.2: Konsonanten und komplexe Konsonantenverbindungen.

kommenden Diphthonge (Papakitsos, 2011, S. 58, Quelle der phonetischen Transkriptionen: [https://de.wikipedia.org/wiki/Altgriechische\\_Phonologie#Diphthonge](https://de.wikipedia.org/wiki/Altgriechische_Phonologie#Diphthonge)). Eine **Ausnahme** bilden die Diphthonge *αι* und *οι*, sofern sie in der letzten Silbe eines deklinierbaren Worts auftreten: Die Silbe gilt dann als kurz (ebd., S. 58). Abweichungen von der allgemeinen Regel können weiterhin durch Auftreten eines Hiats (s. u.) verursacht werden.

3. **Regel  $R_{pl}$  (Positionslänge):** Des Weiteren werden alle Silben, denen zwei oder mehr Konsonanten folgen, lang ausgesprochen. Tabelle 2.2 gibt einen Überblick über die Konsonanten des griechischen Alphabets. Wichtig hierbei ist, dass als „phonologische“ Doppelkonsonanten auch die komplexen Laute ζ (*zeta*), ξ (*xi*) und ψ (*psi*) gelten, denen im geschriebenen Text lediglich ein Buchstabe entspricht: Diese Konsonanten längen mithin auch allein. Eine **Ausnahme** von der Doppelkonsonantenregel bilden Konsonantenfolgen des Typs *mutum cum liquidum* (s. u.): Vor einem *mutum* sind ebenfalls sowohl kurze als auch lange Silben möglich. Papakitsos (ebd., S. 62) führt aus, dass *mutum cum liquidum* am Wortbeginn zur Längung der vorstehenden kurzen Silbe führen kann.
4. **Regel  $R_{zf}$  (Zirkumflex):** Nicht zuletzt sind auch diejenigen Silben lang, die einen durch einen Zirkumflex markierten Vokal oder Diphthong enthalten.

Alle Silben, auf welche die genannten Regeln nicht anwendbar sind, können sowohl kurz als auch lang sein. Die automatische Annotation nicht vollständig durch Regeln modellierbarer Verse ist nur unter Berücksichtigung der bereits erkannten Struktur des Gesamtverses möglich. Erfahrungswerte besagen, dass sich mit Hilfe der genannten Regeln 90 % aller Verse eindeutig (Höflmeier, 1982, S. 7) bestimmen lassen.

Im implementierten Annotationsalgorithmus sind die Annotationsregeln direkt in boolesche Funktionen überführt worden, des Weiteren wurde eine Funktion zur

	Muta	Konsonantenfolge
weich	β	βλ, βρ, βμ, βν
	δ	δλ, δρ, δμ, δν
	γ	γλ, γρ, γμ, γν
hart	π	πλ, πρ, πμ, πν
	τ	τλ, τρ, τμ, τν
	κ	κλ, κρ, κμ, κν
aspiriert	φ	φλ, φρ, φμ, φν
	θ	θλ, θρ, θμ, θν
	χ	χλ, χρ, χμ, χν

TABELLE 2.3: Muta cum liquida.

Modellierung von *muta cum liquida* implementiert. Nähere Ausführungen hierzu finden sich in den Abschnitten 4.4.2.4 und 4.4.2.6.

**Hiat** Das Aufeinandertreffen eines Vokals oder Diphthongs mit einem weiteren Vokal zu einem Hiatt kann unterschiedliche Folgen haben: Beide Vokale (bzw. der Diphthong und der Vokal) können zu einer Länge verschmolzen werden (Synizeise und Krasis). Gleichfalls möglich sind allerdings die Ersetzung des ersten Vokals durch einen Apostroph (Elision) oder die Kürzung des ersten Vokals (Hiatkürzung, Papakitsos, 2011, vgl. auch Leggewie u. a., 1981).

**Muta cum liquida** *Muta cum liquida* sind Konsonantenfolgen mit besonderen metrischen Auswirkungen. Konkret treffen hier ein *mutum*, d. h. ein „stummer“ Konsonant, und ein weiterer Konsonant, ein sogenannter Liquidlaut, aufeinander. Im Griechischen existieren phonologisch harte, weiche und aspirierte, insgesamt neun verschiedene Muta (vgl. Leggewie u. a. (1981, S. 7)). Als *Liquida* kommen die Konsonanten λ (*lambda*), ρ (*rho*), μ (*my*) und ν (*ny*, vgl. Tabelle 2.2) in Betracht. Insgesamt ergeben sich damit die in Tabelle 2.3 aufgeführten 36 Formen von *muta cum liquida*.

**Silbentrennung** Eine automatische Annotation des Versmaßes setzt die automatische Bestimmung der Silbengrenzen im Inneren der annotierten Verse voraus. Regeln für die Silbentrennung lassen sich dem Aufsatz von Papakitsos (2011, S. 58) sowie der Grammatik von Leggewie u. a. (1981, S. 10) entnehmen. Implementierungen finden sich in den in Abschnitt 3.6 genannten Programmen bzw. Bibliotheken sowie in einer weiteren Python-Bibliothek<sup>5</sup>.

## 2.4 Ausgangsdaten

Das in der vorliegenden Arbeit untersuchte Korpus homerischer Texte ist das Ergebnis langjähriger Forschungsarbeiten an der Universität Regensburg (vgl. Höflmeier, 1982, Strasser, 1984). In der hier verwendeten Form wurde es von Kruse (2014, S. 11–22) für die Verarbeitung mit modernen Methoden der Informatik zugänglich gemacht. Das Verdienst Kruses im Hinblick auf die „Dechiffrierung“ der ursprünglich

<sup>5</sup><https://github.com/jtauber/greek-accentuation>.



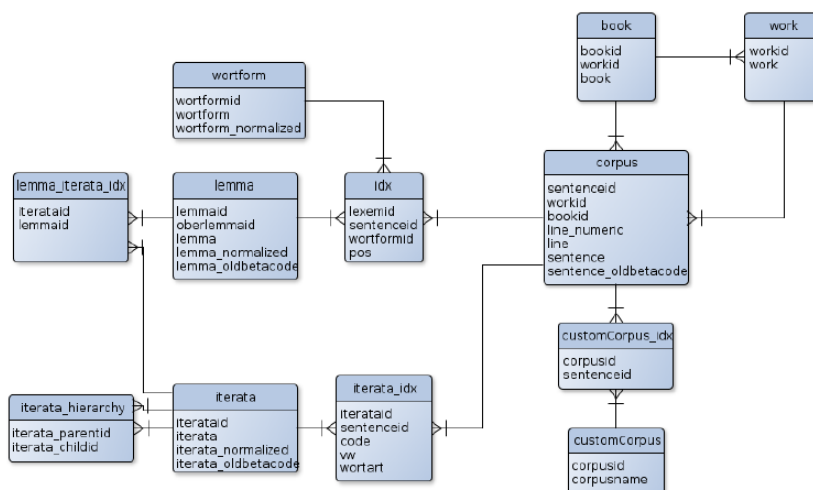


ABBILDUNG 2.3: Schematischer Überblick über die Struktur der von Kruse erzeugten Datenbank (Kruse, 2014, S. 18).

in einem binären, allem Anschein nach undokumentierten Datenformat verwalteten Sprachdaten soll an dieser Stelle noch einmal ausdrücklich hervorgehoben werden! Das von Kruse aufbereitete Korpus enthält neben den Versen zuzüglich Metadaten (z. B. Werk und Buch) auch Listen von Lemmata und sogenannten Iterata, für die Sprache der homerischen Texte typische formelhafte Wendungen. Kruse (ebd.) hat diese Daten in eine leicht abzufragende SQL-Datenbank überführt, deren Struktur in Abbildung 2.3 dargestellt ist. Die Abbildung ist der Arbeit Kruses (ebd., S. 18) entnommen. In der Tabelle „corpus“ enthält die Datenbank 34.750 Verse. Tabelle 2.4 bietet einen Überblick über die jedem Subkorpus zuzuordnende Anzahl von Versen. Die Verse sind sowohl in einem modernen utf8-Format in griechischen Buchstaben als auch in einem sogenannten Betacode-Format hinterlegt. Die zu jedem Vers verwalteten Metadaten umfassen Kürzel<sup>6</sup> und jeweils eine Id für das Werk und das Buch, denen ein Vers zugeordnet werden kann, sowie eine bucheigene Versnummer.

Subkorpus	Kürzel	Id	Anzahl Verse
Ilias	IL	0	15.875
Odyssee	OD	1	12.150
Hesiod, Theogonie	TH	2	1066
Hesiod, Erga	ER	3	840
Ps.-Hesiod, Aspis	AS	4	487
Ps.-Hesiod, Fragmente	HF	5	1835
Homerischer Hymnus	HY	6	2344
Oedipodeia, Oechalias	OE	7	3
Epigonoι	EP	8	1
Cypria	CY	9	51
Aethiopis	AE	10	2
Ilias parva	IP	11	48
Nostoi	NO	12	9
Varia?	V	13	39
			Σ 34.750

TABELLE 2.4: Verteilung der Verse über die einzelnen Subkorpora.

<sup>6</sup>Die Bedeutung des Kürzels V konnte auch in Rücksprache mit Prof. Blößner nicht zweifelsfrei rekonstruiert werden.



## Kapitel 3

# Forschungsstand

Alles prüfe der Mensch, sagen die Himmlischen,  
Daß er, kräftig genährt, danken für Alles lern,  
Und verstehe die Freiheit,  
Aufzubrechen, wohin er will.

Friedrich Hölderlin, aus dem Gedicht „Lebenslauf“

### 3.1 Vorbemerkung

Mit der automatischen Analyse der prosodischen Eigenschaften schriftlich fixierter Texte – zumal von Texten der Lyrik oder aber der hier untersuchten Epoche und Sprache – haben sich nur wenige computerlinguistische Forschungsarbeiten befasst. Aus diesem Grund wird in diesem Abschnitt ein inklusiver Ansatz zum Zusammentragen des relevanten Forschungsstandes gewählt, d. h. es werden nicht nur Studien zur automatischen Versmaßannotation und nicht nur Arbeiten zum Griechischen untersucht, sondern ganz allgemein Arbeiten, die sich mit der Annotation oder Analyse der klanglichen Eigenschaften von Texten beschäftigen. Zu den untersuchten Problemen zählen die automatische Annotation des Wortakzents mit und ohne Berücksichtigung des linguistischen Kontexts (Abschnitt 3.3), die allgemeine Analyse klanglicher Eigenschaften lyrischer Werke (Abschnitt 3.4) und die Annotation des Versmaßes in Werken der Lyrik (Abschnitt 3.5). Nicht zuletzt werden in Abschnitt 3.6 konkrete Implementierungen von Algorithmen zur automatischen Versmaßannotation im Griechischen vorgestellt. Im Anschluss an diesen Überblick leistet Abschnitt 3.7 eine kurze Diskussion und wählt einen Ansatz für die weitere Implementierungsarbeit aus. Zunächst allerdings stellt Abschnitt 3.2 grundlegende Herangehensweisen an die Lösung der genannten Probleme und die computerlinguistische Evaluation der Ergebnisse dar.

### 3.2 Allgemeines zur Methodik und zur Evaluation

In den folgenden Abschnitten werden regelbasierte und stochastische Ansätze sowie Methoden des überwachten maschinellen Lernens (üML) präsentiert. Regelbasierte und üML-Ansätze werden in der Computerlinguistik üblicherweise für die Modellierung von Konzeptlernaufgaben genutzt (für allgemeine Ausführungen zu Regeln und maschinellem Lernen vgl. Beierle, Kern-Isberner und Widera, 2009). Während indes regelbasierte Verfahren versuchen, ein sprachliches Phänomen durch Anwendung explizit formulierter, – deterministischer oder probabilistischer – linguistischer Regeln zu bestimmen, wird beim überwachten maschinellen Lernen der linguistische Sachverhalt in Form von Beispielen beschrieben, d. h. es werden keine

Methode	Regelbasiert	üML	Stochastisch	Beispiel
Finite-State Transducer	X			Reynolds und Tyers (2015, Abschnitt 3.3)
Grammatik	X			Reynolds und Tyers (2015, Abschnitt 3.3)
Conditional Random Fields		X		Estes und Hench (2016, Abschnitt 3.5)
Deep Learning		X		Zabaletak (2017, Abschnitt 3.5)
Hidden-Markov-Modell			X	Zabaletak (2017, Abschnitt 3.5)

TABELLE 3.1: Methodische Einordnung unterschiedlicher Algorithmen.

expliziten Entscheidungsregeln formuliert, vielmehr wird ein geeignetes Klassifikationsmodell im Zuge einer Trainingsphase mit Hilfe eines Trainingsalgorithmus aus den vorhandenen Beispielen erlernt. Dafür wird jede Trainingsinstanz durch ein Tupel repräsentiert, das einerseits eine linguistische Beschreibung des Trainingsdatums (linguistische Merkmale) und andererseits ein Label für die zu erlernende Kategorie enthält. Das im Training erlernte mathematische Modell soll dann in der Lage sein, Tupeln ungesehener Instanzen das Kategorien-Label hinzuzufügen. Stochastische Modelle sind dagegen nicht für das Erlernen abstrakter Konzepte geeignet, vielmehr werden hier mit Hilfe statistischer Zufallsvariablen Wahrscheinlichkeiten für das Eintreten bestimmter Ereignisse berechnet. Unüberwachte Verfahren des maschinellen Lernens wie der von Hench, 2017b (vgl. Abschnitt 3.4), genutzte Clustering-Algorithmus spielen für die hier untersuchte Fragestellung eine geringere Rolle. Tabelle 3.1 ordnet einige der im Folgenden genannten Algorithmen den wesentlichen Kategorien zu.

Für die Evaluation der Qualität automatischer Annotationsalgorithmen existieren unterschiedliche Ansätze. Eine einfache Methode zur Berechnung eines Genauigkeitsmaßes ist das Zählen der korrekt bestimmten Einheiten anhand eines (meist durch manuelle Annotation hergestellten) Goldstandards, d. h. einer Sammlung korrekter Beispiele. Der ermittelte Wert wird dann zur Zahl aller (korrekt oder falsch) annotierten Einheiten ins Verhältnis gesetzt (*accuracy*, vgl. Manning und Schütze, 1999). Allerdings berücksichtigt dieser Evaluationsansatz nicht, für wieviele der zu annotierenden Einheiten überhaupt ein Ergebnis zurückgegeben wurde. Für die Betrachtung dieses Aspekts wurden die Maße Genauigkeit (*precision*,  $p$ ) und Abdeckung (*recall*,  $r$ ) entwickelt. Die Genauigkeit wird bei diesem Evaluationsansatz als Quotient aus der Anzahl der korrekt annotierten Einheiten (*true positives*,  $t_p$ ) und allen überhaupt – korrekt oder falsch (*false positives*,  $f_p$ ) – annotierten Einheiten errechnet, vgl. Definition 1 (*precision* und *accuracy* sind also identisch). Der Abdeckungswert  $r$  gibt dagegen an, welcher Anteil aller Einheiten überhaupt korrekt annotiert wurde (*true positives*), die *false negatives* ( $f_n$ ) bezeichnen dabei diejenigen Einheiten, für die der Algorithmus kein Versmaß bestimmen konnte. Die entsprechende Gleichung findet sich in Definition 2 (ebd., S. 268 f.).

**Definition 1 (Genauigkeit)**  $p = \frac{t_p}{t_p + f_p}$

**Definition 2 (Abdeckung)**  $r = \frac{t_p}{t_p + f_n}$

Zusätzlich zu diesen beiden Maßen kann als globales, einheitliches Evaluationsmaß noch das F-Maß (*F measure*,  $F$ ) aus Genauigkeit und Abdeckung entsprechend Definition 3 bestimmt werden (ebd., S. 269):

**Definition 3 (F-Maß, allgemeine Definition)**  $F = \frac{1}{\alpha \frac{1}{p} + (1-\alpha) \frac{1}{r}}$

Hierbei ist  $\alpha$  ein Parameter für die flexible Gewichtung von Abdeckung und Präzision. Werden beide Größen gleich gewichtet ( $\alpha = 0.5$ ), kann Gleichung 3 wie folgt vereinfacht werden (ebd.):

**Definition 4 (F-Maß, vereinfachte Definition)**  $F = \frac{2*pr}{p+r}$

Als Bezugsgrößen für die Berechnung der genannten Evaluationsmaße kommen im hier betrachteten Fall sowohl ganze Verse als auch einzelne Silben in Betracht. Über die Evaluation eines einzelnen Annotationsergebnisses mit Hilfe der genannten Metriken hinaus ist allerdings auch der Vergleich unterschiedlicher Annotationsergebnisse von Interesse. Beispielsweise kann es für praktische Belange notwendig sein, ein Maß für die Übereinstimmung zwischen unterschiedlichen Annotationen zu errechnen: Eine hohe Übereinstimmung ist dabei ein Indiz für zuverlässige Annotationsergebnisse, wohingegen eine schwache Übereinstimmung auf eine mangelhafte konzeptuelle Modellierung des Problems hindeutet, d. h. bei mindestens einem der in den Vergleich einbezogenen – menschlichen oder maschinellen – Annotatoren lässt sich eine mangelnde Stringenz der Annotation beobachten. In diesem Fall sollten die zwischen beiden Annotatoren erhobenen Übereinstimmungen mit Vorsicht betrachtet werden. Liegt dagegen eine hohe Übereinstimmung zwischen beiden Annotatoren vor, gilt dies als Indiz für eine korrekte Konzeptualisierung des annotierten Sachverhalts, die Übereinstimmungen können dann auch ohne Evaluation des gesamten Datensatzes als Hinweise auf die Korrektheit der Annotation verstanden werden. Als Maß für die Übereinstimmung zwischen zwei Annotatoren hat sich in der Computerlinguistik der Kappa-Koeffizient eingebürgert, der in Gleichung 5 definiert wird (Cohen, 1960, alle Ausführungen hierzu nach Artstein und Poesio, 2008, S. 556–562):

**Definition 5 (Kappa)**  $\kappa = \frac{A_o - A_e}{1 - A_e}$

Der Koeffizient  $A_o$  bezeichnet hierbei die zwischen den Annotatoren *beobachtete* Übereinstimmung, d. h. den Anteil der gleich annotierten Fälle an der Stichprobe,  $A_e$  dagegen die *erwartete* Übereinstimmung. Der Ausdruck  $1 - A_e$  gibt an, welches Maß an Übereinstimmung über zufällig erwartbare Gleichheit hinaus (maximal) erreicht werden kann. Der Ausdruck  $A_o - A_e$  dagegen bezeichnet das tatsächlich beobachtete Maß nicht zufälliger Übereinstimmung. Der Berechnung des Koeffizienten  $A_e$  liegt die Annahme zugrunde, dass bei zufälliger Annotation für die Annotatoren unterschiedliche Ergebnisverteilungen entstehen würden.  $A_e$  errechnet sich mithin (für zwei Annotatoren  $c_1$  und  $c_2$ ) wie folgt:

$$A_e^k = \sum_{k \in K} \hat{P}(k|c_1) * \hat{P}(k|c_2) = \sum_{k \in K} \frac{n_{c_1k}}{i} * \frac{n_{c_2k}}{i} = \frac{1}{i^2} \sum_{k \in K} n_{c_1k} n_{c_2k} \quad (3.1)$$

In Gleichung 3.1 bezeichnet  $K$  die Menge der annotierten Kategorien (für die Hexameterannotation gilt also  $|K| = 32$ ). Die Wahrscheinlichkeiten  $P(k|c_1)$  und  $P(k|c_2)$  werden dann auf Grundlage der tatsächlichen Annotationen geschätzt. Der Ausdruck  $n_{c_1k}$  gibt folglich die Anzahl der von Annotator  $c_1$  der Kategorie  $k \in K$  zugeordneten Beobachtungen an,  $i$  bezeichnet die Stichprobengröße.

Der  $\kappa$ -Koeffizient kann Werte zwischen 0 und 1 annehmen. Ein Wert nahe 0 bezeichnet dabei zufällige Übereinstimmung, höhere Werte, insbesondere  $\kappa > 0,8$ , weisen auf eine systematische Übereinstimmung zwischen den Datensätzen hin<sup>1</sup>.

### 3.3 Automatische Annotation des Wortakzents

Die Annotation des Wortakzents kann insofern als ein der Versmaßannotation ähnliches Problem aufgefasst werden, als es hier um die Zuweisung eines binären Merkmals (hier: betont/unbetont) zu linguistischen Einheiten (hier: Wörter) geht. Darüber hinaus bauen einige der noch zu referierenden Ansätze zur Versmaßannotation sehr stark auf Informationen über Betonungsmuster auf (vgl. Abschnitt 3.5). Aus diesen Gründen sollen hier zwei einfache Ansätze für die automatische Bestimmung des Wortakzents kurz zusammengefasst werden.

Ciobanu, Dinu und Dinu (2014) beschreiben einen überwachten Ansatz zum Erlernen des primären Wortakzents im Rumänischen ohne Berücksichtigung des linguistischen Kontexts. Zunächst heben die Autoren hervor, dass das Rumänische eine morphologisch reiche Sprache sei, in der einem einzigen Lemma eine Vielzahl unterschiedlicher Wortformen mit kaum vorhersehbarem Wortakzent zugeordnet werde. Ciobanu et al. modellieren das Problem als sequentielles Tagging-Problem: Dabei werden die Buchstaben eines Wortes nacheinander – von links nach rechts – durch das erlernte Tagging-Modell evaluiert; der als Träger des Hauptakzents identifizierte Buchstabe wird dabei mit einer speziellen Markierung (einem *Tag*) versehen. Die linguistischen Merkmale, auf deren Grundlage das Tagging-Modell erlernt wird, umfassen (ebd., S. 65):

- Buchstabensequenzen (*character n-grams*) in Nachbarschaft des aktuell evaluierten Buchstabens,
- Verallgemeinerte Buchstabensequenzen, die anstelle von Informationen zu konkreten Buchstaben nur Informationen über Vokale und Konsonanten enthalten,
- Informationen zur Position des aktuell untersuchten Buchstabens im Wort.

Des Weiteren trainieren Ciobanu et al. ein Modell für die Silbentrennung im Rumänischen. Die beiden Modelle werden nacheinander auf ein Wort angewandt, um eine komplette Analyse zu erzeugen. In einem Evaluationsexperiment wurde eine Klassifikationsgenauigkeit (*accuracy*) von 97 % erzielt.

Der von Reynolds und Tyers (2015) genutzte Ansatz zur Annotation des Wortakzents im Russischen – einer Sprache mit variablem Wortakzent und einer Vielzahl morphologisch und semantisch ambiger Wortformen, die ohne Einbezug semantischer und kontextueller Informationen allein anhand ihrer Betonung unterschieden werden können – beruht auf endlichen Automaten. Konkret generiert zunächst ein Finite-State Transducer die Menge aller für ein gegebenes Wort möglichen morphologischen Analysen. In einem zweiten Schritt wird dann eine Grammatik genutzt, um unter Einbezug syntaktischer Informationen aus dieser Menge die korrekte Lesart auszuwählen, der dann ein Wortakzent zugewiesen werden kann. Anders als der Ansatz von Ciobanu, Dinu und Dinu (2014) annotiert der Algorithmus von Reynolds und Tyers mithin die Wortbetonung im Kontext eines konkreten Satzes.

<sup>1</sup>Artstein und Poesio (2008) liefern allerdings eine ausführliche und kontroverse Diskussion zur Festlegung von Akzeptanzgrenzen für Übereinstimmungskoeffizienten.

Reynolds und Tyers evaluieren ihren Annotationsalgorithmus auf einem mit etwa 7700 Wörtern äußerst kleinen Korpus von Übungstexten für Russischlernende. Von den im Korpus enthaltenen Wörtern werden nur die knapp 4000 mehrsilbigen Wörter zur Evaluation herangezogen. Mit der besten Variante ihres Algorithmus können die Autoren 96 % der Testwörter korrekt annotieren. Ohne Einbezug der Grammatik zur Disambiguierung des syntaktischen Kontexts können im günstigsten Setting 90 % der Testwörter korrekt annotiert werden, wobei nur noch für Wörter ohne betonungsrelevante Ambiguitäten eine Ausgabe erfolgt.

### 3.4 Untersuchung klanglicher Eigenschaften lyrischer Werke

In diesem Abschnitt werden einige Studien vorgestellt, die computerlinguistische Methoden nutzen, um philologische Fragestellungen zu bearbeiten. Der Überblick soll dabei nicht nur verdeutlichen, welche Forschungsfragen aktuell mit Hilfe moderner Computertechnik bearbeitet werden können, sondern auch die Breite des infrage kommenden Methodeninventars unterstreichen.

McCurdy, Srikumar und Meyer (2015) befassen sich mit der Analyse von Reimen in Werken der Lyrik. Konkret wird ein Formalismus entwickelt, mit dessen Hilfe Reime unterschiedlichen Typs in lyrischen Werken identifiziert werden können. Dafür wird jedes Wort im Gedicht zunächst in ein Reimobjekt („*rhyming object*“) zerlegt, das einerseits eine Repräsentation der phonetischen Wortgestalt und andererseits seine Oberflächenstruktur (Buchstabenfolge) enthält. Des Weiteren wird eine sehr detaillierte Analyse der Silbenstruktur jedes einzelnen Wortes erarbeitet. Auf Grundlage dieser Analyse können alsdann sehr genaue Regeln („*rhyming templates*“) für die Beschreibung unterschiedlicher Reimtypen angegeben werden, die sowohl die visuellen und strukturellen (Buchstaben) als auch die klanglichen und phonetischen Bestandteile von Reimen berücksichtigen. Im verbleibenden Teil ihres Artikels präsentieren die Autoren die Open-Source-Software RhymeDesign<sup>2</sup>, welche die aufgestellten Regeln nutzt, um Reime in Gedichten zu identifizieren. Gleichfalls haben die Nutzer die Möglichkeit, eigene Reimregeln zu definieren und für die Analyse von Gedichten zu nutzen. Offen bleibt, auf welche natürlichen Sprachen die gefundenen Reimregeln überhaupt angewendet werden können. Der Umstand, dass alle im Artikel gegebenen Beispiele dem Englischen entstammen, legt indes nahe, dass die Analyse auf diese Sprache beschränkt bleibt.

Hench, 2017a, widmet sich den klanglichen Eigenschaften („*soundscapes*“) mittelhochdeutscher Minnesang-Dichtung. Konkret wird ein – noch recht explorativer – Ansatz zum Verständnis von Wirkung und Bedeutung phonologischer Eigenschaften einzelner Gedichte und deren Bezug zum lyrischen Ich präsentiert. Der Autor stellt die Hypothese auf, dass der nur durch Rezitation zutage tretende klangliche Charakter der untersuchten lyrischen Werke wesentlich für deren Verständnis sei und diskutiert eine Reihe bekannter und im Fach zum Teil kontrovers diskutierter Aspekte aus phonologischem Blickwinkel, wobei ein besonderer Schwerpunkt auf die Verteilung offener und geschlossener Silben sowohl in einem konkreten Gedicht als auch in einem Korpus von Gedichten gelegt wird. Da die Voraussetzung für eine solche Diskussion die Verfügbarkeit eines Programms zur automatischen Silbentrennung in allen Gedichten des Korpus ist, geht Hench auch kurz auf

<sup>2</sup><http://www.sci.utah.edu/~nmccurdy/rhymeDesign/>.



einen Silbentrennungs-Algorithmus ein, mit dem das Untersuchungskorpus bearbeitet wurde. Dieser Algorithmus basiert auf linguistischen Prinzipien der Silbentrennung und erreicht, so Hench, auf einem Testdatensatz von 1000 Wörtern eine Annotationsgenauigkeit von 99 %. In Hench, 2017b, präsentiert der Autor schließlich einen einfachen textbasierten Clustering-Algorithmus, mit dessen Hilfe klanglich „ähnliche“ Texte in einem Textkorpus erkannt werden sollen.

Einen explorativen Ansatz zur klanglichen Analyse moderner und postmoderner Gedichte, die nicht mit Hilfe strenger (regelbasierter) Metrik-Schemata beschrieben werden können, präsentieren Baumann und Meyer-Sickendiek (2016). Die Autoren nutzen für ihre Studie Audio-Aufnahmen von Lesungen der untersuchten Gedichte, und zwar einerseits Aufnahmen der Autoren selbst und poetisch naive („*poetically naïve*“) maschinengenerierte Aufnahmen andererseits. Aus beiden Typen von Aufnahmen werden klangliche Merkmale wie beispielsweise die Tonhöhe extrahiert. Die Analyse von Verteilungen dieser Merkmale in den untersuchten Datensätzen soll alsdann Aufschluss darüber geben, welche Merkmale die gesprochene poetische Rede (zusätzlich zu einem möglicherweise vorhandenen Versmaß) von der gewöhnlichen Alltagsrede unterscheiden. Dieser Ansatz wird in der Arbeit von Meyer-Sickendiek, Hussein und Baumann (2017) fortgeführt. Es werden deutsche und englische Gedichte untersucht.

### 3.5 Automatische Versmaßannotation

Dieser Abschnitt widmet sich schließlich den eigentlichen Untersuchungen zur automatischen Versmaßannotation in lyrischen Texten. Dabei werden Forschungsarbeiten zu einer Reihe unterschiedlicher Sprachen betrachtet, denn lediglich zwei Studien (Pavese und Boschetti, 2003, Papakitsos, 2011) beschäftigen sich ausschließlich mit der in dieser Arbeit interessierenden Sprache und Varietät: dem Griechisch der homerischen Epoche.

Die mehrbändige Arbeit von Pavese und Boschetti (2003) widmet sich der Aufbereitung und Edition aller formelhaften Ausdrücke (*formula*, ähnlich wie die bereits in Abschnitt 2.4 erwähnten Iterata sind formelhafte Ausdrücke wiederkehrende Wendungen im epischen Text) in den Epen Homers. Zu den für die Formeledition nötigen Vorbereitungsarbeiten gehört dabei auch eine prosodische Analyse. Diese Analyse wird in mehreren Schritten semi-automatisch mit Hilfe regulärer Ausdrücke und linguistischer Regeln durchgeführt. So werden die Buchstaben eines Verses zunächst nach ihrem prosodischen Lautwert kodiert, beispielsweise werden lange Vokale durch den Buchstaben H (*longa vel diphthongus*) und kurze Vokale durch den Buchstaben E (*brevis*) ersetzt (ebd., S. 84). Im Korpus mehrfach auftretende Strings mit nicht eindeutig bestimmbar Vokalen (im ersten Schritt ersetzt durch A: *incerta vocalis*) werden alsdann manuell bearbeitet und korrigiert, den Vokalen wird also händisch ein Lautwert (lang oder kurz) zugewiesen.

Die eigentliche metrische Analyse wird im Anschluss in zwei Schritten als Kombination aus einer silbenweisen Analyse, welche die einzelnen Silben isoliert von einander betrachtet, und einer Positionsanalyse, bei der die Silben im Kontext des Gesamtverses gesehen werden, durchgeführt. In diesem Sinne ähnelt das Vorgehen dem von Höflmeier (1982, vgl. Abschnitt 3.6). Bei der silbenweisen Analyse versuchen Pavese und Boschetti (2003), Buchstabensequenzen – konkreten Verbindungen

aus Konsonanten und Vokalen – einen prosodischen Wert zuzuweisen. Die Zuweisung geschieht wiederum durch Ersetzung der Buchstabensequenz durch ein entsprechendes Kodierungselement. Die Positionsanalyse ist als eine Serie von Ersetzungen mit Hilfe regulärer Ausdrücke implementiert. Im Ergebnis wird (ebd., S. 75) mehr als 90 % der analysierten Verse ein Versmaß zugewiesen, allerdings wird die Korrektheit der Zuweisungen nicht evaluiert.

Greene, Bodrumlu und Knight (2010) befassen sich in ihrem Aufsatz nicht nur mit der Analyse, sondern auch mit der automatischen Erzeugung und Übersetzung lyrischer Texte. Konkret werden Shakespeare-Sonette bearbeitet. Als Schwerpunkt der Analyse wird die metrische Analyse gewählt, die Grundbegriffe der metrischen Analyse werden – in etwas eigenwilliger Art („Iambic is a common meter that sounds like da-DUM da-DUM da-DUM etc.“) – kurz erläutert; die im Artikel gewählte Notation für eine betonte Silbe lautet  $S^*$ , die für eine unbetonte Silbe  $S$ .

Die Autoren nutzen zunächst einen gewichteten Finite-State Transducer (FST), der Wortsequenzen – Verse – wortweise auf Betonungsmuster der Länge 1–4 abbildet, also  $S, S^*, S - S$ , etc. Insgesamt ergeben sich somit 32 mögliche Ausgabesequenzen, denen im ersten Schritt identische Wahrscheinlichkeiten zugewiesen werden, d. h.  $p = \frac{1}{32}$  für jede Ausgabesequenz. Danach wird der Transducer trainiert, d. h. Tupel, bestehend aus einem Vers der Shakespeare-Sonette (insgesamt 17.134 Wörter) und der dem gewünschten Versmaß entsprechenden Ausgabesequenz werden genutzt, um die Wahrscheinlichkeiten an den Zustandsübergängen des Finite-State Transducers korrekt zu gewichten. Noch fehlende Varianten und Annotationsfehler im Versmaß werden durch die Erweiterung des Trainingsdatensatzes und die Kombination des Haupt-Transducers mit einem zweiten, einfacheren Transducer abgefangen. Die Transducer werden mit Hilfe der Open-Source-Software „Carmel“<sup>3</sup> implementiert. Auf diese Weise können 81 % der analysierten Testverse – bei einem Testdatensatz von gerade einmal 70 Versen – korrekt annotiert werden. Die in der Transducer-Kaskade kodierten Wahrscheinlichkeiten für die Abbildung von Wörtern auf Betonungsmuster werden in der Folge noch für weitere Aufgaben genutzt, etwa für die Erzeugung von Versen mit einem gewünschten Versmaß. Auf diese Aspekte soll hier nicht weiter eingegangen werden, aufgrund ihrer Originalität werden aber in Abbildung 3.1 einige der von Greene, Bodrumlu und Knight (ebd., S. 529) automatisch erzeugten englischen Gedichte dargestellt.

Papakitsos (2011) beschreibt ein regelbasiertes System für die Versmaßannotation in den Epen Homers, der Ilias und der Odyssee. Dieses System prüft in strukturierter Weise das Vorliegen konkreter Zuordnungsbedingungen. Zunächst wird im Vers eine Silbentrennung vorgenommen. Verse mit 12 bzw. 17 Silben werden dann standardmäßig den Verstypen 50 bzw. 00 laut Abbildung 2.2 zugeordnet. Die weitere Analyse wird silbenweise durchgeführt. Eine Besonderheit dabei ist, dass Wissen über die Häufigkeit des Auftretens von Spondeen in konkreten Kontexten in die Verarbeitung einbezogen wird: Falls in einem bestimmten Versfuß häufiger ein Spondeus auftritt als in anderen Versfüßen, wird zuerst diese Möglichkeit überprüft (ebd., S. 61 ff.). Konkret bedeutet dies, dass ein Vers nicht silbenweise von links nach rechts verarbeitet wird, vielmehr beginnt die Verarbeitung bei den „leichten“ Silben (ebd., S. 64). Einbezogen werden des Weiteren Informationen über die Anzahl der Silben im Vers: So muss ein Vers mit 13 Silben mindestens einen Daktylus enthalten (vgl. Abbildung 2.2). Ist dieser sicher gefunden, ist keine weitere Analyse mehr nötig, das zeichenweise Durcharbeiten des gesamten Verses wie etwa bei Höflmeier (1982, vgl. Abschnitt 3.6) kann entfallen. Der Autor evaluiert seinen Algorithmus im

<sup>3</sup><https://github.com/graehl/carmel>.

The women of the night  
 Again and all the way  
 Like a mouse in the white  
 Not the heart of the day.  
 - - -  
 Of the bed to trust me  
 Around her twists the string  
 But i will not tell thee  
 Fire changes everything.  
 - - -  
 A son of the right hand confines  
 His uncle could have broken in  
 Towards the high bank and the pines  
 Upon the eyes and i have been  
 - - -  
 Into one of her hundred year old  
 Or the house in a house in a cold  
 The first time she met him  
 Like a mouse in the dim  
 For me to the moon and when i told  
 - - -  
 Into one of them some years before  
 His own man or the house in a more  
 The moon and when the day  
 Into one of the way  
 With the breath from the first time  
 she swore

ABBILDUNG 3.1: Beispiele für automatisch mit Finit-State Transducern generierte englische Gedichte aus der Studie von Greene, Bodrumlu und Knight (2010, S. 529.)

Hinblick auf Annotationsgenauigkeit und -abdeckung auf 700 Testversen aus der Ilias. Es werden eine Präzision von 80 % und eine Abdeckung von 98 % (ebd., S. 67) erreicht.

Navarro-Colorado (2015) setzt sich mit der Analyse spanischer Sonette des 16. und 17. Jahrhunderts auseinander. Das Untersuchungskorpus umfasst 5078 Sonette, die Texte liegen in einem XML-Format vor. Zunächst hebt Navarro-Colorado als besondere Schwierigkeit hervor, dass orthographische Silben nicht direkt auf metrische Silben abbildbar seien: „[T]here is not a direct relationship between linguistic (sic!) and metrical syllables ...“. Des Weiteren beschreibt er einige Besonderheiten des Spanischen im Hinblick auf die metrische Analyse: Besondere Komplexitäten in Form von Ambiguitäten, d. h. der Existenz mehrerer korrekter Lösungen, sieht der Autor im Umgang mit Vokalen (Verschmelzung von Vokalen unterschiedlicher orthographischer Silben zu einer metrischen Silbe oder umgekehrt Verteilung eines Vokals auf zwei Silben).

Navarro-Colorado nutzt linguistische Regeln, um Wörter in (orthographische) Silben zu teilen. Alsdann wird ein Wortarten-Tagger verwendet, um metrische Silben zu identifizieren, wobei allerdings nicht deutlich ist, inwiefern ein Wortarten-Tagger – also ein Computer-Programm, das jedem Wort eine Wortart zuweist – für diese Aufgabe nutzbar ist. Lediglich vermuten lässt sich, dass hier Regeln genutzt werden, denen zufolge nur Instanzen bestimmter Wortarten (z. B. Nomina) Träger eines metrischen Akzents sein können. In einem letzten Schritt wendet Navarro-Colorado einen weiteren Satz von Regeln zur Behandlung komplexer Vokalfolgen an. In Fällen, in denen für einen Vers mehrere metrische Lesarten existieren, wird



das im Korpus am häufigsten auftretende Muster gewählt. Im Rest des Artikels beschreibt der Autor die semantische Analyse seines Korpus von Sonetten; auf diesen Aspekt wird hier nicht mehr eingegangen.

Estes und Hench (2016) präsentieren einen Ansatz für die automatische Versmaßannotation in Texten der mittelhochdeutschen Epik. Diese werden zunächst im Vergleich zu den Texten der lateinischen Epik als in metrischer Hinsicht hybrid charakterisiert, da bei der Realisierung der Versfüße trotz der eigentlich auf dem Wechsel betonter und unbetonter Silben beruhenden Metrik Einflüsse aus der quantifizierenden Metrik der griechischen und römischen Epik (vgl. Abschnitt 2.3.1) zutage treten. Aus diesem Grund sind bei der Versmaßannotation im Mittelhochdeutschen sowohl die Silbenlänge – gemessen als halbe, ganze oder doppelte More – als auch die Wortbetonung zu annotieren. Dabei können einer Silbe die folgenden Wertkombinationen zugewiesen werden (ebd. S. 3):

- einmorig, mit primärem Wortakzent
- einmorig, mit sekundärem Wortakzent
- einmorig, unbetont
- halborig (kurz), mit primärem Wortakzent
- halborig, mit sekundärem Wortakzent
- halborig, unbetont
- zweimorig (lang und betont)
- Elision

Da Estes und Hench für die automatische Versmaßannotation einen überwachten Lernalgorithmus nutzen, werden Trainingsdaten benötigt. Insgesamt haben die Autoren 825 mittelhochdeutsche Verse aus den Epen „Der arme Heinrich“, „Parzival“ und „Wigalois“ manuell annotiert, von denen 10 % in der Folge für Evaluationszwecke zurückgehalten werden. Der Rest der Verse wird zum Trainieren des Klassifikationsalgorithmus verwendet. Vor dem Beginn der Versmaßannotation wurde zudem unter Nutzung linguistischer Regeln ein Silbentrennungs-Algorithmus implementiert, der – evaluiert auf den ersten 1000 Wörtern des „Iwein“ von Hartmann von Aue – eine Annotationsgenauigkeit von ca. 99 % erzielte (ebd., S. 4).

Auf Grundlage dieser Daten wird ein Klassifikationsmodell erstellt, als Algorithmus werden *Conditional Random Fields* (CRF) genutzt. Die linguistischen Merkmale, anhand derer Klassifikationsentscheidungen getroffen werden, sind die folgenden (ebd., S. 4. f.):

- Position der Silbe im Vers
- Silbenlänge in Buchstaben
- Unterschiedliche *character n-grams* für jede Silbe (vgl. den in Abschnitt 3.3 referierten Ansatz von Ciobanu, Dinu und Dinu, 2014)
- Vorliegen einer Elision

- Vorliegen von linguistischen Indikatoren für das regelhafte Auftreten von Längen und Kürzen
- Position der Silbe im Hinblick auf eine Wortgrenze

Das erstellte Klassifikationsmodell weist jeder Silbe eine der oben aufgeführten acht Wertkombinationen (Äquivalenzklassen) zu. Es wird eine silbenweise Evaluation durchgeführt, d. h. die Präzision und Abdeckung werden auf die Gesamtmenge der verarbeiteten Silben bezogen. Als globales Evaluationsmaß über der Vereinigungsmenge aller Äquivalenzklassen geben die Autoren ein F-Maß von 0,904 auf den Testdaten an. Eine feinkörnige Analyse der Ergebnisse zeigt wenig überraschend, dass hochfrequente Klassen am besten erkannt werden können.

Agirrezabal u. a. (2016) nutzen einen Finite-State-Transducer, um eine Versmaßannotation in englischen Gedichten durchzuführen. Da dieser Ansatz – für das Englische angemessen, für das quantifizierende Versmaß des Griechischen jedoch irrelevant – weitgehend auf der Analyse von Wortbetonungssequenzen beruht, soll er hier allerdings nicht im Detail referiert werden. Die Evaluation wird sowohl in Bezug auf Verszeilen (759 englische Verse wurden analysiert) als auch auf Silben (7076 Silben) durchgeführt. Die versweise Genauigkeit des Algorithmus beträgt lediglich 26 %. Die silbenweise Genauigkeit beträgt dagegen 87 %.

Zabaletak (2017) geht ähnlich wie Agirrezabal u. a. zunächst vom Englischen aus und konzentriert sich auf die Annotation des Versmaßes anhand der Analyse von lexikalischen Betonungssequenzen. Wortbetonungen können in diesem Ansatz in einem Wörterbuch nachgeschlagen werden, wobei freilich Sonderfälle und das Auftreten von nicht im Wörterbuch auffindbaren Wörtern behandelt werden müssen. Neben dem Englischen werden zudem das Spanische und das Baskische analysiert. Des Weiteren werden Experimente mit überwachten Lernverfahren unter Nutzung unterschiedlicher Algorithmen durchgeführt. Die hierfür verwendeten Merkmale lauten wie folgt:

- Position der Silbe im Wort und im Vers
- Anzahl der Silben im Vers
- phonologische Eigenschaften der Silbe
- Wortlänge
- Buchstabensequenzen (die bereits erwähnten *character n-grams*) am Wortende

Die Annotationsgenauigkeit dieser Verfahren bleibt indes hinter der Performanz des bereits von Agirrezabal u. a. präsentierten regelbasierten Ansatzes zurück (ebd., S. 98). Allerdings kann die Hinzunahme von Kontextinformationen (Silben, Wörter, Wortarten und Betonungsmuster im Umfeld des aktuell untersuchten Worts) in das linguistische Modell die Genauigkeit deutlich steigern. In der besten Konfiguration erreicht der Autor mit diesem Verfahren eine silbenweise Klassifikationsgenauigkeit von 89 % und eine versweise Genauigkeit von 41 %.

Des Weiteren führt Zabaletak (ebd., S. 99) eine zweite Reihe von Experimenten mit Algorithmen durch, mit deren Hilfe Sequenzen erlernt werden können. Konkret werden Hidden-Markov-Modelle und *Conditional Random Fields* trainiert. Im besten Setting können hier mit *Conditional Random Fields* und den genannten lokalen und kontextuellen Merkmalen weitere Qualitätssteigerungen (91 % silbenweise Genauigkeit und 51 % versweise Genauigkeit) erzielt werden. Nicht zuletzt experimentiert

Zabaletak (ebd., S. 101 ff.) mit aktuellen Deep-Learning-Verfahren. Das beste Ergebnis wird erreicht, indem ein neuronales Netz mit einem *Conditional Random Field* kombiniert wird: Dabei erlernt das neuronale Netz eine phonologische Repräsentation der Eingabedaten, auf deren Grundlage der Klassifizierer alsdann die wahrscheinlichste Ausgabesequenz, also das Versmaß, errechnet. Die Annotationsgenauigkeit wird so noch einmal auf 95 % (silbenweise) bzw. 71 % (versweise) gesteigert. Im Spanischen werden noch höhere Genauigkeiten erreicht, im Baskischen dagegen niedrigere. Im Anschluss an diese Diskussion präsentiert Zabaletak (ebd., S. 109 ff.) Ergebnisse von Experimenten mit unüberwachten Lernverfahren, die aber durchgängig niedriger sind als die zuvor genannten Werte.

### 3.6 Implementierungen

Als Grundlage für die Entwicklung eines eigenen Annotationssystems stehen neben den in den letzten Abschnitten ausführlich referierten Ansätzen aus der computerlinguistischen Forschung mehrere auf das Griechische ausgerichtete Implementierungen zur Verfügung, die nun kurz vorgestellt werden sollen. Ein mittlerweile historisch zu nennender Ansatz für die automatische Versmaßannotation in gerade den hier untersuchten Texten der frühgriechischen Epik ist zunächst der Arbeit und dem dazugehörigen Pascal-Skript von Höflmeier (1982) zu entnehmen. Ziel der Arbeit Höflmeiers ist eine vollständige metrische Analyse mit folgender quantitativer Auswertung der Versmaßverteilungen. Die Feststellung Höflmeiers (ebd., S. 4), dass mit Hilfe seines Verfahrens „... nahezu 30 000 Verse in wenigen Stunden bearbeitet werden“ konnten, mag dabei aus heutiger Sicht zunächst humoristisch anmuten – in Anbetracht des mit manueller Annotation verbundenen Aufwands stellt sie allerdings auch heute noch einen großen Fortschritt dar. Nicht nur im Hinblick auf die von heutigen Anforderungen zur damaligen Zeit deutlich abweichenden Computersysteme bleibt die Arbeit von Höflmeier durchaus bemerkenswert. Höflmeier (ebd., S. 6) implementiert einen regelbasierten, zweistufigen Ansatz für die Versmaßanalyse:

„Wie bei jedem Programm mußte auch hier ein Algorithmus gefunden werden, der möglichst unkompliziert und zugleich höchst effizient ist. Ein Minimum metrischer Gesetzmäßigkeiten sollte eine maximale Anzahl von Quantitäten festlegen lassen, d. h. vorerst wurden zur Vereinfachung nicht alle metrischen Regeln einprogrammiert. Die dadurch entstehenden Fehler und ungelösten Fälle erfuhren ihre Korrektur in einem zweiten Schritt durch die Ermittlung des Sitzes der Silbe im Vers.“

Höflmeier (ebd., S. 6 ff.) referiert einige der in Abschnitt 2.3.3 zusammengefassten linguistischen Regeln für die Versmaßannotation und führt das in Abbildung 2.2 wiedergegebene Schema für die 32 Hexameter-Varianten an. Im Anschluss daran wird, wie bereits festgestellt, eine recht umfangreiche Analyse von Versmaßverteilungen durchgeführt.

Das von Höflmeier entwickelte Pascal-Programm ist mit linguistischem Expertenwissen gesättigt, allerdings mit seinen über 2000 Code-Zeilen auch recht unübersichtlich. Die Software liest Korpusverse aus einer Datendatei mit festgelegtem Format. Ein Korpusvers wird dabei als Zeichen-Array aufgefasst und für die zeichenweise Verarbeitung werden sechs Zeiger auf Array-Positionen verwendet. Das Programm ist weitgehend undokumentiert. Linguistische Regeln sind in Form

einzelner Prozeduren implementiert, die durch ein mehrstufiges System von Meta-Prozeduren zu größeren Verarbeitungsschritten zusammengefasst werden, was die Verständlichkeit des Codes weiter beeinträchtigt. Insgesamt enthält das Programm 39 Prozeduren.

Die eigentliche Analyse wird in der Meta-Prozedur *Analyze\_Metre* durchgeführt, die von der höherstufigen Meta-Prozedur „Verarbeiten“ aufgerufen wird. Die Trennung der einzelnen Silben wird durch vokalweises Vorrücken der Zeiger realisiert. In einer Unterprozedur *Evaluate\_Syllable* wird alsdann die Anwendbarkeit kontextunabhängig geltender Annotationsregeln für die aktuell untersuchte Silbe geprüft, das Ergebnis wird mit Hilfe einer booleschen Variable zwischengespeichert (Länge sicher bestimmt oder nicht). Falls die Länge nicht eindeutig bestimmt werden kann, wird eine dreistufige Kaskade linguistischer Regeln angewandt. Nicht zuletzt existieren Prozeduren für die Interaktion mit dem Benutzer des Programms, der die Annotation eines Verses überprüfen und ggf. (beispielsweise immer bei Vorliegen eines Hiats) korrigieren kann. Höflmeier führt keine Evaluation seines Algorithmus durch.

Eine Open-Source-Implementierung für die automatische Annotation des Versmaßes in griechischen Texten ist in der Python-Bibliothek „Classical Language Toolkit“ (CLTK) enthalten<sup>4</sup>. Laut Dokumentation<sup>5</sup> bietet das CLTK computerlinguistische Werkzeuge für die Verarbeitung einer Vielzahl „klassischer Sprachen“, darunter Alt-Ägyptisch, Alt- und Mittelenglisch, Mittelhochdeutsch, Altnorwegisch, Sanskrit, Tocharisch B – und Griechisch. In Bezug auf die von der Bibliothek abgedeckte Varietät bzw. das Entwicklungsstadium des Griechischen wird keine Angabe gemacht.

Für die CLTK-Implementierung existiert keine ausführliche Dokumentation, der (im Vergleich zu dem Skript von Höflmeier deutlich kompaktere) Quellcode für die Versmaßannotation ist jedoch online einsehbar<sup>6</sup> und kann daher analysiert werden. Zunächst werden einige – typische – Vorverarbeitungsschritte ausgeführt, welche das weitere Vorgehen erleichtern sollen. So werden nicht-alphabetische und Interpunktionszeichen entfernt, alle Großbuchstaben werden in kleine Buchstaben umgewandelt, Wort- und Silbengrenzen werden erkannt. Des Weiteren wird ein Großteil der Akzente aus dem Text entfernt. Als dann werden auf jede Silbe ähnlich wie in dem Pascal-Skript von Höflmeier einige linguistische Regeln angewandt:

- *Long by nature*: Die Regel erkennt Silben, die einen Diphthong oder naturlangen Vokal enthalten.
- *Long by position*: Die Regel erkennt positionsbedingte Längungen, wenn etwa die nächste Silbe mit einem Doppelkonsonanten beginnt (Ausnahme: *muta cum liquida*) oder an der Silbengrenze zwei Konsonanten aufeinander treffen.

Insgesamt scheint es sich mithin um einen allgemeinen Annotationsalgorithmus, d. h. nicht um einen auf die Hexameter-Analyse zugeschnittenen Algorithmus zu handeln. Die Silbentrennung wird regelbasiert durchgeführt. Der Algorithmus legt dabei im Wesentlichen fest, dass eine Silbe auf einen Vokal oder einen Konsonanten enden kann. Eine vergleichbare Implementierung findet sich auch in der Python-Bibliothek der Altphilologin Anna Conser<sup>7</sup>.

<sup>4</sup><https://github.com/cltk>.

<sup>5</sup><http://docs.cltk.org/en/latest/>.

<sup>6</sup><https://github.com/cltk/cltk/blob/master/cltk/prosody/greek/scanner.py>.

<sup>7</sup>[https://github.com/aconser/greek\\_scansion](https://github.com/aconser/greek_scansion).

Artikel	Methode	Testdaten	Evaluation	Beste Performanz
Pavese und Boschetti, 2003	Regeln	–	–	–
Greene, Bodrumlu und Knight, 2010	gewichteter FST	70 Verse	Annotationsgenauigkeit	0.81
Papakitsos, 2011	Regeln	700 Verse	Präzision Abdeckung	$p = 0.80$ $r = 0.98$
Navarro-Colorado 2015	Regeln, Tagger	–	–	–
Estes und Hench, 2016	CRF	75 Verse	F-Maß	$F = 0.90$
Agirrezabal u. a., 2016	FST	759 Verse	Annotationsgenauigkeit	versweise: 0.26 silbenweise: 0.87
Zabaletak, 2017	Deep Learning	78 Gedichte (Englisch)	Annotationsgenauigkeit	versweise: 0.71 silbenweise: 0.95
Höflmeier, 1982	Regeln	–	–	–
CLTK	Regeln	–	–	–
A. Conser	Regeln	–	–	–
H. Ranker	Regeln, gewichteter FST	–	–	–

TABELLE 3.2: Zusammenfassende Übersicht über die referierten Ansätze zur automatischen Versmaßannotation.

Eine weitere Open-Source-Implementierung konkret für die Hexameter-Annotation bietet die Python-Bibliothek Hexameter von Hope Ranker<sup>8</sup>. Dieser Algorithmus nutzt sowohl lokale linguistische Regeln zur Natur- bzw. Positionslänge von Vokalen als auch gewichtete endliche Automaten zur Auflösung von Ambiguitäten beispielsweise im Zusammenhang mit *muta cum liquida*. Dabei wird für jeden der zu analysierenden sechs Versfüße ein eigener Automat eingesetzt. Seltene bzw. unerwünschte Phänomene sind mit hohen Gewichten belegt, bei der Auswahl einer gültigen Analyse aus der Menge der möglichen Analysen wird schließlich die Variante mit dem niedrigsten Gewicht gewählt.

### 3.7 Diskussion und Auswahl eines Implementierungsansatzes

Die vorstehenden Ausführungen haben deutlich gemacht, welche philologischen Fragestellungen im Hinblick auf die prosodischen Eigenschaften von Gedichten von Interesse sind. Des Weiteren ist deutlich geworden, dass für die Lösung prosodischer Probleme unterschiedliche Herangehensweisen gewählt werden: Das Spektrum genutzter Technologien reicht von allgemein regelbasierten Ansätzen über endliche Automaten bis hin zu überwachten und unüberwachten Methoden des maschinellen Lernens.

Tabelle 3.2 zeigt eine zusammenfassende Übersicht über die in den vorstehenden Abschnitten vorgestellten Arbeiten und Implementierungen. Die Übersicht zeigt, dass im Vergleich zu den in den Abschnitten 3.3 und 3.4 referierten Arbeiten regelbasierte Ansätze in Form von endlichen Automaten oder anderen Implementierungen deutlich überwiegen – lediglich Estes und Hench (2016) und Zabaletak (2017) präsentieren üML-Ergebnisse. Allerdings evaluieren etwa Estes und Hench auf gerade einmal 75 Versen, so dass durchaus fraglich ist, ob sich das von ihnen berichtete Ergebnis auf größere Datensätze verallgemeinern lässt.

Die in dieser Arbeit untersuchten Hexameterverse der frühgriechischen Epik – die Behauptung lässt sich leicht durch Angabe des allgemeinen Hexameterschemas aus

<sup>8</sup><https://github.com/epilanthanomai/hexameter>.

Abbildung 2.1 beweisen – können als reguläre Sprache aufgefasst werden. Endliche Automaten sind mithin auf die hier interessierende Problemstellung anwendbar. Wird zudem bedacht, dass es sich bei einem konkreten Versmaß um eine linguistische Struktur mit vergleichsweise wenigen Variationsmöglichkeiten handelt – im Fall des Hexameters existieren 32 unterschiedliche Varianten, vgl. Abschnitt 2.3.1 und Abbildung 2.2 – und dass sich in vielen Fällen konkrete Werte anhand weniger lokaler Kontextparameter bestimmen lassen, erscheint die Wahl eines auf endlichen Automaten beruhenden Ansatzes als durchaus berechtigt. Dies unterscheidet das Problem der Versmaßannotation beispielsweise von dem in Abschnitt 3.3 kurz vorgestellten Problem der Annotation des Wortakzents: In Bezug auf dieses hatten etwa Reynolds und Tyers (2015) detailliert dargestellt, dass für die kontextsensitive Annotation der Wortbetonung im Russischen sowohl syntaktische als auch – idealerweise – semantische Informationen nicht-lokaler Natur ausgewertet werden müssen. Auch die Schwierigkeit der Beschaffung geeigneter Trainingsdaten könnte ein Grund für das Überwiegen symbolischer Herangehensweisen an die Versmaßannotation sein. Im Hinblick auf die in dieser Studie zu bearbeitende Sprache und Varietät waren zudem in den Abschnitten 2.2 und 2.3.2 Merkmale hervorgehoben worden, welche die Nutzung von üML-Methoden erschweren können, nämlich einerseits die formenreiche Morphologie des Griechischen und das Nebeneinander von Elementen unterschiedlicher Entwicklungsstadien in den Texten Homers: Beide Phänomene würden den Umfang des zu modellierenden Vokabulars deutlich erhöhen. Wird schließlich nach der Performanz der diskutierten Ansätze gefragt, macht der Vergleich der referierten Arbeiten deutlich, dass – ungeachtet der Unmöglichkeit, die vorgetragenen Performanzwerte direkt zueinander in Bezug zu setzen – in nahezu allen Fällen eine zufriedenstellende Performanz erreicht wird, d. h. wenigstens für das reguläre Gros der Verse werden korrekte Annotationen erzeugt. Dabei wird zwischen einer silben- und einer versweisen Evaluation unterschieden.

Im folgenden Abschnitt wird ein regelbasierter Ansatz für die Hexameter-Annotation in den Texten der frühgriechischen Epik entwickelt. Konkret werden die in Abschnitt 2.3.3 vorgestellten Regeln in endliche Automaten eingebettet, welche die Verarbeitung der zu analysieren Verse steuern. Damit wird nicht nur den soeben vorgetragenen Argumenten, sondern auch der in Abschnitt 1 aufgestellten Anforderung einer expliziten Regelanwendung Rechnung getragen. Weitere Vorteile ergeben sich daraus, dass endliche Automaten sich leicht visualisieren lassen, so dass eine auch für Linguisten nachvollziehbare Dokumentation des wissensbasierten Annotationsansatzes möglich erscheint, sowie aus den mathematisch wohldefinierten Eigenschaften endlicher Automaten.



## Kapitel 4

# Automatische Versmaßannotation

... Denn nie, sterblichen Meistern gleich,  
Habt ihr Himmlischen, ihr Alleserhaltenden,  
Daß ich wüßte, mit Vorsicht  
Mich des ebenen Pfades geführt.

Friedrich Hölderlin, aus dem Gedicht „Lebenslauf“

### 4.1 Vorbemerkung

In diesem Kapitel wird ein Algorithmus zur automatischen Annotation von griechischen Hexameterversen mit Hilfe endlicher Automaten vorgestellt. Dafür wird zunächst in Abschnitt 4.2 ein kurzer Abriss der Geschichte endlicher Automaten in der Computerlinguistik gegeben. Alsdann werden in Abschnitt 4.3 die für die wissenschaftliche Diskussion des Themas unabdingbaren Begriffe definiert<sup>1</sup>. Abschnitt 4.4 schließlich enthält die Beschreibung des Annotationsprogramms. Abschnitt 4.5 fasst dieses Kapitel zusammen.

### 4.2 Nutzung endlicher Automaten für computerlinguistische Fragestellungen

Endliche Automaten gehören zu den grundlegenden Konzepten der theoretischen Informatik (vgl. Schulz, 2016). Gleichzeitig gehören endliche Automaten zu den etablierten Konzepten der Computerlinguistik. Während die Nutzbarkeit endlicher Automaten zunächst im Zusammenhang mit der computergestützten Modellierung phonologischer Probleme wissenschaftlich erforscht wurde, etablierten sich endliche Automaten insbesondere seit den achtziger Jahren als Werkzeuge für die regelbasierte Bearbeitung komplexer morphologischer Fragestellungen (Koskenniemi, 1983, vgl. auch die online verfügbare Überblicksdarstellung von Karttunen und Beesley<sup>2</sup>). Nicht zuletzt trug in dieser Phase auch die Entwicklung breit nutzbarer Programmierwerkzeuge (Beesley und Karttunen, 2003) zur Popularität endlicher Automaten in der Computerlinguistik bei.

<sup>1</sup>Bei den definierten Konzepten – deterministischen endlichen Automaten und Finite-State Transducern – handelt es sich um Grundbegriffe der Informatik. Die Definitionen dienen mithin im Wesentlichen der Vollständigkeit der Darstellung und der Lektüre-Erleichterung. Auf eine formal vollständige Herleitung der Definitionen sowie auf Beweise der aufgestellten Behauptungen wird unter Verweis auf die zitierten Quellen aber verzichtet.

<sup>2</sup><http://www.ling.helsinki.fi/~koskenni/esslli-2001-karttunen/>.

In späteren Entwicklungsphasen wurden endliche Automaten in einem breiten Spektrum computerlinguistischer Teildisziplinen wie etwa der Spracherkennung (*speech recognition*), der Wortartenannotation (*part-of-speech tagging*), der syntaktischen Analyse (*parsing*) und der Informationsextraktion (*information extraction*) erprobt<sup>3</sup>. Einen Überblick über entsprechende Forschungsarbeiten bieten beispielsweise Roche und Schabes (1997). Nach der Ablösung vieler regelbasierter computerlinguistischer Ansätze durch statistische Verfahren und Methoden des maschinellen Lernens haben sich endliche Automaten insbesondere in der morphologischen Analyse agglutinierender, d. h. besonders formenreicher Sprachen wie dem Finnischen und dem Türkischen (Oflazer, 1994) bewährt. Neuere Forschungsarbeiten (Yli-Jyrä, Karttunen und Karhumäki, 2006) belegen das beachtliche Maß an Komplexität, das entsprechende Verfahren mittlerweile erreicht haben; häufig werden nicht-indoeuropäische Sprachen mit nur wenigen Sprechern und einem Mangel an computerlinguistisch verwertbaren Ressourcen untersucht, für die regelbasierte Ansätze oft die einzig nutzbaren Methoden liefern (Tyers u. a., 2017).

### 4.3 Definitionen

#### 4.3.1 Deterministischer endlicher Automat

Ein deterministischer endlicher Automat (DEA)  $M$  lässt sich als Computerprogramm auffassen, das Eingabewörter zeichenweise verarbeitet und für jedes Eingabewort  $w$  entscheidet, ob es zu der von  $M$  erkannten regulären Sprache  $L_M$  gehört oder nicht. Dafür nutzt der Automat eine Menge  $Q$  unterschiedlicher Zustände. Endet die Verarbeitung des letzten eingegebenen Zeichens in einem akzeptierenden Zustand von  $M$ , akzeptiert der Automat das Eingabewort:  $w \in L_M$ . Andernfalls verwirft der Automat das Eingabewort:  $w \notin L_M$ . Formal wird der Begriff des endlichen deterministischen Automaten wie folgt definiert (nach Schulz, 2016, KE 2, S. 2):

**Definition 6 (Deterministischer endlicher Automat)** Ein deterministischer endlicher Automat  $M$  wird durch ein Tupel  $(Q, \Sigma, \delta, q_0, F)$  dargestellt. Hierbei ist

- $Q$  eine endliche, nicht-leere Menge, die Zustandsmenge,
- $\Sigma$  ein endliches Alphabet,
- $\delta$  eine Funktion  $\delta : Q \times \Sigma \rightarrow Q$ , die Übergangsfunktion,
- $q_0$  eine Element aus  $Q$ , der Startzustand,
- $F$  eine Teilmenge von  $Q$ , die Menge der akzeptierenden Zustände.

DEAs lassen sich graphisch als Zustandsdiagramme darstellen. Abbildung 4.1 ist Schulz (2016, KE 2, S. 5) entnommen und stellt das Zustandsdiagramm eines einfachen DEA dar:  $q_0$  ist der Startzustand,  $q_2$  ist der einzige akzeptierende Zustand (doppelt umrandet). An den Kanten sind die Buchstaben des Eingabealphabets abgetragen. Dabei bezeichnen die gerichteten Kanten den Übergang von einem Ausgangszustand zu einem Zielzustand bei Eingabe des an der Kante dargestellten Buchstaben. Der dargestellte DEA akzeptiert u. a. die Wörter  $w_1 = bab, w_2 = baabbb$ , nicht

<sup>3</sup>Die englischen Bezeichnungen werden hier aufgrund ihrer weiteren Verbreitung und ihrer Eindeutigkeit angeführt. Beispielsweise entsprechen dem deutschen Terminus „Spracherkennung“ die englischen Termini „language recognition“ bzw. „language identification“ und „speech recognition“.



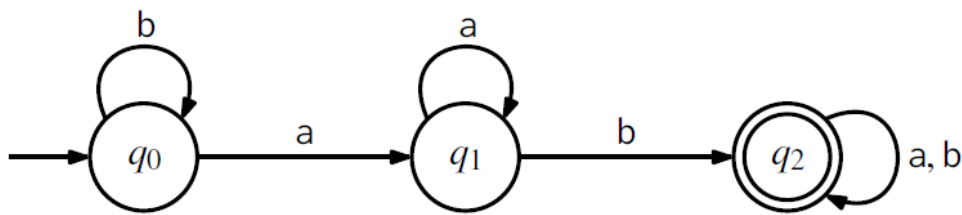


ABBILDUNG 4.1: Zustandsdiagramm eines einfachen DEA (Schulz, 2016, KE 2, S. 5).

aber das Wort  $u = b$ , das heißt  $w_1, w_2 \in L_M, u \notin L_M$ . Die reguläre Sprache  $L_M$  ist dabei gerade die Menge der von  $M$  akzeptierten Wörter. Der Automat ist deterministisch, da die Übergangsfunktion für jedes Tupel  $(q \in Q, a \in \Sigma)$  den zu erreichenden Folgezustand eindeutig festlegt. DEAs können mithin als deterministische Variante endlicher – deterministischer und nicht-deterministischer – Automaten aufgefasst werden. Eine allgemeine Definition endlicher Automaten findet sich in Roche und Schabes (1997, S. 4). Eine Definition nicht-deterministischer endlicher Automaten findet sich in Schulz (2016).

Aufgrund der Abschlusseigenschaften regulärer Sprachen sind unterschiedliche Operationen über Mengen von DEAs definiert. Hierzu zählen Komplement, Vereinigung, Konkatenation, Schnitt und Kleene-Stern (ebd.).

### 4.3.2 Finite-State-Transducer

#### 4.3.2.1 Definition

Ähnlich wie ein DEA akzeptiert oder verwirft ein Finite-State Transducer (FST)  $F$  Eingabewörter. Zusätzlich jedoch wird zu jedem eingegebenen Zeichen  $a \in \Sigma_1$  ein Zeichen  $b \in \Sigma_2$  ausgegeben. Formal wird der Begriff des Finite-State-Transducers wie folgt definiert (nach Roche und Schabes, 1997b, S. 14):

**Definition 7 (Finite-State Transducer)** Ein Finite-State Transducer  $F$  wird durch ein Tupel  $(\Sigma_1, \Sigma_2, Q, q_0, F, E)$  dargestellt. Hierbei ist

- $\Sigma_1$  ein endliches Eingabealphabet,
- $\Sigma_2$  ein endliches Ausgabealphabet,
- $Q$  eine endliche, nicht-leere Zustandsmenge,
- $q_0 \in Q$  der Startzustand,
- $F \subseteq Q$  die Menge der akzeptierenden Zustände,
- $E \subseteq Q \times \Sigma_1^* \times \Sigma_2^* \times Q$  die Menge der Kanten, welche die Zustände miteinander verbinden.

Abbildung 4.2 stellt einen einfachen Finite-State Transducer dar. An den Kanten sind vor den Doppelpunkten die Buchstaben des Eingabealphabets abgetragen. Hinter den Doppelpunkten stehen die dem Zustandsübergang entsprechenden Buchstaben des Ausgabealphabets. Die Darstellung macht deutlich, dass Finite-State Transducer

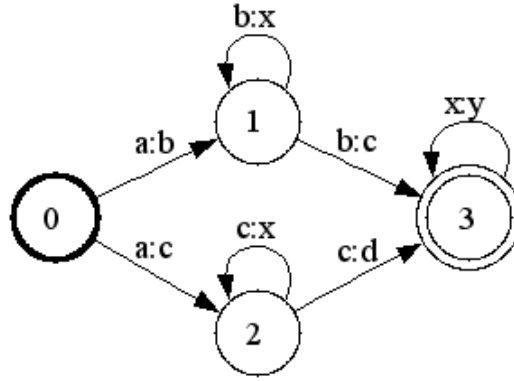


ABBILDUNG 4.2: Finite-State Transducer (Quelle: [https://de.wikipedia.org/wiki/Transduktor\\_\(Informatik\)](https://de.wikipedia.org/wiki/Transduktor_(Informatik))).

nicht deterministisch sind<sup>4</sup>. Beispielsweise kann der FST aus Abbildung 4.2 bei Eingabe des Zeichens  $c$  aus dem Zustand 2 direkt in den akzeptierenden Zustand 3 übergehen und dabei ein  $d$  ausgeben ( $(2, c, d, 3) \in E$ ). Alternativ kann ein Buchstabe  $x$  ausgegeben werden, dabei bleibt der Transducer im Zustand 2 ( $(2, c, x, 2) \in E$ ).

Deutlich wird außerdem, dass ein Finite-State Transducer im Unterschied zu einem DEA nicht nur Eingabewörter verwerfen oder akzeptieren kann, sondern auch eine Relation zwischen Wörtern  $w_1 \in \Sigma_1^*$  und  $w_2 \in \Sigma_2^*$  definiert. Zur Herleitung dieser Relation definieren Roche und Schabes (ebd., S. 16) zunächst den Begriff der erweiterten Kantenmenge:

**Definition 8 (Erweiterte Kantenmenge eines FSTs)** Die erweiterte Kantenmenge  $\hat{E}$  eines Finite-State Transducers ist die kleinste Teilmenge  $Q \times \Sigma_1^* \times \Sigma_2^* \times Q$ , so dass:

- (i)  $\forall q \in Q : (q, \epsilon, \epsilon, q) \in \hat{E}$
- (ii)  $\forall w_1 \in \Sigma_1^*, \forall w_2 \in \Sigma_2^* : (q_1, w_1, w_2, q_2) \in \hat{E} \wedge (q_2, a, b, q_3) \in E \implies (q_1, w_1 a, w_2 b, q_3) \in \hat{E}$

Mit Hilfe der erweiterten Kantenmenge kann beschrieben werden, welchen Ausgabewörtern die Eingabewörter durch den Transducer zugeordnet werden und in welchem Zustand dieser Prozess endet. Aufbauend auf dieser Definition kann jedem Finite-State Transducer nun eine Relation  $L(T) : \Sigma_1^* \times \Sigma_2^*$  zugeordnet werden (ebd.):

$$L(T) = \left\{ (w_1, w_2) \in \Sigma_1^* \times \Sigma_2^* \mid \exists (i, w_1, w_2, q) \in \hat{E}, q \in F \right\} \quad (4.1)$$

$L(T)$  ordnet jedem durch den FST akzeptierten  $w_1 \in \Sigma_1^*$  ein  $w_2 \in \Sigma_2^*$  zu. Bei Bedarf können die Kanten eines nicht-deterministischen Finite-State Transducers mit Gewichten versehen werden.

#### 4.3.2.2 Operationen und Anwendung

Finite-State Transducer sind unter Komplement und Vereinigung abgeschlossen (ebd., S. 17). Für bestimmte Typen von Transducern postulieren Roche und Schabes zudem den Abschluss unter Komposition und Schnitt. Relevant sind diese Aspekte für die Anwendung der Automaten auf computerlinguistische Probleme. Roche

<sup>4</sup>Die Determinisierbarkeit von FSTs wird aber von Roche und Schabes (1997) ausführlich diskutiert.

und Schabes (ebd., S. 21–25) beschreiben jeweils ein Beispiel für die Lösung eines (einfachen) computerlinguistischen Problems durch Komposition und Schnitt von Finite-State Transducern. So wird im ersten Fall – Regelkomposition – eine Kaskade von Finite-State Transducern eingesetzt, wobei zunächst derjenige Transducer angewendet wird, der die allgemeinste Regel kodiert. Die auf die so erzeugte Ausgabe aufbauenden Transducer kodieren alsdann zunehmend speziellere Regeln. Im zweiten Fall – Regelschnitt – bildet die Schnittmenge der Ausgaben der einzelnen Transducer die Lösungsmenge.

Der Anwendung von Finite-State Transducern auf Eingabewörter widmen Roches und Schabes (ebd., S. 41–43) einen eigenen Abschnitt:

„One way of computing the output of a given input consists in traversing the transducer in all ways that are compatible with the current input symbol performing backtracking if necessary until a complete path is found. ... A more natural way to compute the output of a transducer for a given input, is to view the input as an FSA<sup>5</sup>. On this method, the application of the transducer to the input is computed as a kind of intersection of the transducer with the automaton.“

Für FSTs, die als Kantenlabel ausschließlich nicht-leere Einzelbuchstaben erhalten können, definieren die Autoren (ebd., S. 43) die Schnittoperation wie folgt:

**Definition 9 (Schnitt eines FSTs und eines endlichen Automaten)** Sei  $T_1$  ein Finite-State Transducer mit  $T_1 = (\Sigma, \Sigma_1, Q_1, i, F_1, E_1)$ , so dass  $E_1 \subseteq Q_1 \times \Sigma_1 \times \Sigma_2^* \times Q_1$ . Sei weiterhin  $A_2$  ein endlicher Automat mit  $A_2 = (\Sigma, Q_2, i_2, F_2, E_2)$ , so dass  $E_2 \subseteq Q_2 \times \Sigma_1 \times Q_2$ . Der Schnitt aus  $T_1$  und  $A_2$  ist dann definiert als endlicher Automat  $A = (\Sigma, Q_1 \times Q_2, (i_1, i_2), F_1 \times F_2, E)$  mit  $E \subseteq (Q_1 \times Q_2) \times \Sigma_2^* \times (Q_1 \times Q_2)$ , so dass

$$E = \bigcup_{(q_1, a, b, r_1) \in E_1, (q_2, a, r_2) \in E_2} ((q_1, q_2), b, (r_1, r_2)) \quad (4.2)$$

Die Zustandsmenge des entstehenden endlichen Automaten entspricht dem Kreuzprodukt  $Q_1 \times Q_2$ . Die neue Kantenmenge ist die Vereinigung der Übergänge zwischen den neu entstandenen kombinierten Zuständen, wobei als Kantenlabel lediglich die Ausgaben des ursprünglichen FSTs genutzt werden.

## 4.4 Annotationsprogramm

### 4.4.1 Allgemeiner Annotationsalgorithmus

Der implementierte Annotationsalgorithmus ist mehrschrittig, da er – ähnlich wie bereits die Algorithmen von Höflmeier (1982) sowie Pavese und Boschetti (2003) – eine lokale Suche mit Hilfe linguistischer Regeln mit einer globalen Analyse des Versmaßes kombiniert. Die lokale Suche orientiert sich dabei an dem Algorithmus von Papakitsos (2011), der bereits in Abschnitt 3.5 referiert wurde. Insgesamt wird das Problem in die folgenden Teilschritte zerlegt:

1. **Vorverarbeitung und Silbentrennung:** Vor dem Beginn der eigentlichen Annotation wird für jeden Vers eine Vorverarbeitung durchgeführt. Diese umfasst

<sup>5</sup>Die englische Abkürzung *FSA* (*finite-state automaton*) bezeichnet einen – nicht notwendigerweise deterministischen – endlichen Automaten.

im Wesentlichen das Entfernen nicht benötigter Diakritika und Interpunktionszeichen sowie die Überführung der Groß- in Kleinbuchstaben. Diese Schritte vereinfachen die Suche nach langen Silben in Schritt 2, bei der für die linguistische Regelprüfung viele String-Vergleiche durchgeführt werden müssen. Dann wird der Gesamtvers in Silben zerlegt. Die Qualität der Silbentrennung ist maßgeblich für die folgenden Verarbeitungsschritte.

2. **Lokale Suche:** Ausgehend von der im ersten Schritt bestimmten Silbenzahl wird eine bestimmte Anzahl von Spondeen im Vers gesucht. Als „gefunden“ werden dabei nur sicher lange Silben (entsprechend den lokalen Regeln aus Abschnitt 2.3.3) betrachtet, ein Spondeus setzt das Aufeinanderfolgen zweier langer Silben an einer bestimmten Versstelle voraus. Ist die für die ermittelte Silbenzahl benötigte Anzahl an Spondeen gefunden, terminiert der Algorithmus für den Eingabervers. Andernfalls wird Schritt 3 ausgeführt, dafür wird diesem ein teilannotierter Vers übergeben. Die korrekte Durchführung von Schritt 2 (und ggf. den Schritten 3 und 4) wird durch hierarchische endliche Automaten (vgl. Abschnitte 4.4.2.6 und 4.4.2.7) kontrolliert.
3. **Globale Analyse:** Mit Hilfe eines Finite-State Transducers wird versucht, dem teilannotierten Vers aus Schritt 2 das korrekte Versmaß zuzuweisen. Dem als Eingabewort fungierenden teilannotierten Vers soll dabei ein korrektes Hexameterschema als Ausgabewort zugeordnet werden.
4. **Fehlerbehandlung:** Falls auch der Finite-State Transducer aus Schritt 3 das Hexameterschema nicht bestimmen kann, wird eine Fallback-Routine ausgeführt.

Abbildung 4.3 stellt den Annotationsalgorithmus als Pseudo-Code dar, der hier zum besseren Verständnis eingehend erläutert werden soll. Zunächst werden in Zeile 2 fünf hierarchische DEAs für die unterschiedlichen Verslängen (13–16 Silben) initialisiert: *hfsa13*, *hfsa14*, *hfsa15* und *hfsa16* (vgl. Abschnitt 4.4.2.7); der DEA *simple* wird für die Fehlerbehandlung bei längeren oder kürzeren Versen genutzt. Ebenfalls initialisiert wird ein *preprocessor* für die Vorverarbeitung der Verse und die Zerlegung in Silben. Alsdann wird der Algorithmus für jeden Vers einzeln ausgeführt (*for*-Schleife von Zeile 4 bis Zeile 45):

1. **Vorverarbeitung und Silbentrennung:** In den Zeilen 5 und 6 werden Funktionen der *preprocessor*-Klasse (vgl. Abschnitt 4.4.2.3) genutzt, um Vorverarbeitung und Silbentrennung durchzuführen. Ist die Silbenzahl im Vers bekannt, beginnt die Analyse getrennt für die unterschiedlichen Versklassen (*if-else*-Kaskade auf Grundlage der Silbenzahl).
2. **Lokale Suche:**
  - (a) Zunächst wird überprüft, ob überhaupt eine Analyse durchzuführen ist. Verse mit weniger als 12 oder mehr als 17 Silben können eigentlich keine gültigen Hexameter sein. Allerdings wird in Betracht gezogen, dass das Ergebnis der Silbentrennung fehlerhaft sein kann. Hat der Vers mindestens 9 Silben, wird dennoch eine Analyse mit Hilfe des DEA *simple* durchgeführt. Hierfür wird der Automat zunächst zurückgesetzt, ihm werden der Verstext und die ermittelten Silben übergeben (Zeilen 9–11). Die Analyse (Zeile 12) entspricht der Fehlerbehandlung in den übrigen Fällen.
  - (b) Für Verse mit genau 12 bzw. 17 Silben existiert dagegen nur eine korrekte Hexametervariante (vgl. Abbildung 2.2), die sofort zwischengespeichert

```

1 Annotationsalgorithmus Hexameter()
   Input : Korpus Korpus mit griechischen Hexameter-Versen
   Output: Verse mit Versmaßannotation

2   Initialisiere DEAs: hfsa13, hfsa14, hfsa15, hfsa16, simple;
3   Initialisiere preprocessor, initialisiere Versmaß-Objekt als leeren String;

4   for Vers in Korpus do
5       Text ← preprocessor.vereinfache(Vers);
6       Silben ← preprocessor.trenne(Text);
7       if len(Silben) < 12 or > 17 then // Irreguläre Verse
8           if len(Silben) > 8 then // Hexameter ab 9 erkannten Silben
9               simple.warte();
10              simple.Vers.Silben ← Silben;
11              simple.Vers.Text ← Text;
12              simple.analysiere();
13              if (Versmaß gefunden) then Versmaß ← simple.Vers.Versmaß;
14          end
15          else output Warnung;
16      end
17      else if len(Silben) == 12 then // Vers besteht nur aus Spondeen
18          Versmaß ← '-----X';
19          simple.warte();
20          simple.Vers.Silben ← Silben;
21          simple.Vers.Text ← Text;
22          if not simple.istRichtig(Versmaß) then
23              simple.analysiere();
24              if (Versmaß gefunden) then Versmaß ← simple.Vers.Versmaß;
25          end
26      end
27      else if len(Silben) == 17 then
28          // Behandlung analog zum vorhergehenden Fall
29      end
30      else
31          switch len(Silben) do
32              case 13 do
33                  hfsa13.warte();
34                  hfsa13.Vers.Silben ← Silben;
35                  hfsa13.Vers.Text ← Text;
36                  hfsa13.analysiere();
37                  if hfsa13.Zustand == 'kein Spondeus gefunden' then
38                      hfsa13.vervollständige(); // Globale Analyse
39                      // ggf. Durchführung einer Fehlerbehandlung
40                  end
41                  // ggf. Speicherung des Versmaßes im Versmaß-Objekt
42              end
43              ...; // Für 14, 15 und 16 Silben analog
44          end
45      end
46      if (Versmaß-Objekt ist leer) then Versmaß ← 'NOT RESOLVED';
47      output Versmaß;
48  end

```

ABBILDUNG 4.3: Allgemeiner Annotationsalgorithmus.

(Zeile 18) und auch ausgegeben werden kann, sofern eine Plausibilitätsprüfung (Zeile 22) ein positives Ergebnis erbringt. Andernfalls wird wieder die Fehlerbehandlung angestoßen (Zeile 23).

- (c) In allen anderen Fällen wird ein dedizierter Analyseprozess angestoßen. Dieser Prozess einschließlich einer möglichen Vervollständigung und Fehlerbehandlung in den Schritten 3 und 4 wird von einem auf Verse der gegebenen Länge spezialisierten hierarchischen DEA (im Pseudo-Code bearbeitet der DEA *hfsa13* Verse mit 13 Silben) gesteuert. Auch der DEA wird zunächst zurückgesetzt (Zeile 32), alsdann werden ihm die Ergebnisse von Vorverarbeitung und Silbentrennung übergeben (Zeilen 33–34). Sodann wird in Zeile 35 die Spondeensuche mit Hilfe lokaler linguistischer Regeln angestoßen. Kann der DEA dabei nicht die gesuchten vier Spondeen (für Verse mit 13 Silben, exemplarische Erläuterung) finden, wird eine teilannotierte Version des Versmaßes erstellt und der DEA wechselt in den Zustand *Globale Analyse* (Zeilen 36–38), die Schritte 3 und ggf. 4 werden ausgeführt.
- 3. **Globale Analyse:** Der Finite-State Transducer (vgl. Abschnitt 4.4.2.8), mit dessen Hilfe die Analyse durchgeführt wird, ist ein Attribut des DEAs aus Schritt 2 und wird im Zustand *Globale Analyse* von diesem aufgerufen. Zunächst wird allerdings ein teilannotierter Vers erstellt, d. h. auch die Länge derjenigen Silben, die während der lokalen Suche gar nicht explizit betrachtet wurden, wird mit Hilfe der linguistischen Regeln bewertet. Der Transducer nutzt dann das Hexameterschema, um allen Silben einen Wert zuzuweisen.
- 4. **Fehlerbehandlung:** Die Fehlerbehandlung wird von den DEAs überwiegend im Ergebnis einer fehlschlagenden Plausibilitätsprüfung aufgerufen<sup>6</sup>. Für den einfachen DEA *simple* dagegen stellt die Routine für die Fehlerbehandlung das Standardvorgehen dar, sie wird also im Zustand *analysiere* des DEA *simple* direkt aufgerufen (vgl. z. B. Zeile 12 in Abbildung 4.3). Für alle anderen DEAs wird die Fehlerbehandlung nur dann aufgerufen, wenn sowohl die lokale Suche als auch die globale Analyse kein regelkonformes Ergebnis erbracht haben. Der Algorithmus der Prozedur *korrigiere*, welche die Fehlerbehandlung implementiert, ist in Abbildung 4.4 als Pseudo-Code dargestellt. Zunächst wird in Zeile 2 ähnlich wie bei Höflmeier (1982) eine buchstabenweise Bestimmung von Längen und Kürzen durchgeführt<sup>7</sup>. So trägt der Algorithmus dem Umstand Rechnung, dass fehlerhafte Annotationsergebnisse durch eine falsche Silbentrennung verursacht werden können. Der vorannotierte String wird dann zur Vervollständigung dem FST übergeben (Zeile 3). Falls ein Ergebnis gefunden werden kann, wird dieses gespeichert, die Prozedur terminiert, löst aber vorher ggf. einen Zustandsübergang des DEA aus (wenn sich dieser nicht bereits im letzten Zustand befindet, Zeilen 13–15). Kann auch an dieser Stelle keine Lösung gefunden werden, sucht der Algorithmus nach möglichen Korrekturen des Versmaßes an Stellen, an denen ein Hiatus vorliegt. Aktuell werden nur mögliche Synizesen behandelt (Zeilen 5–8, zu anderen Möglichkeiten vgl. Abschnitt 2.3.3). Mit Hilfe eines regulären Ausdrucks werden durch Synizeze zusammenfassbare Vokalfolgen identifiziert. Da diese Vokale bei Synizeze

<sup>6</sup>In Abbildung 4.3 entspricht der Plausibilitätsprüfung die Funktion *istRichtig()*, die beispielsweise in Zeile 22 aufgerufen wird. Die Plausibilitätsprüfung wird aber auch – im Pseudo-Code nicht sichtbar – nach der lokalen und globalen Suche aufgerufen.

<sup>7</sup>Die Prozedur arbeitet direkt auf den Attributen der Klasse *verse* (vgl. Abschnitt 4.4.2.5), so dass keine Ein- oder Ausgabeparameter benötigt werden.

zu einer langen Silbe verschmelzen, wird die Vokalfolge durch *einen* naturlangen Vokal ersetzt (Zeile 6). Alsdann wird der Fehlerbehandlungsprozess durch einen rekursiven Prozeduraufruf mit den aktualisierten Daten erneut gestartet (Zeile 7). Falls auch dieser Korrekturschritt fehlschlägt, ist das Versmaß nicht bestimmbar. In Zeile 10 wird dann ein entsprechender Parameter des DEA gesetzt, der den Übergang in einen Misserfolgzustand veranlasst.

```

1 Prozedur korrigiere()
  // buchstabenweise Analyse von Längen und Kürzen
2  self.analysiereBuchstabenweise();
  // teilannotiertes Versmaß durch FST vervollständigen lassen
3  self.verwendeTransducer();
4  if (FST konnte den String nicht auflösen) then
5    if (Synizese könnte vorliegen) then
6      // Ersetze Vokalfolge durch einen langen Vokal
      self.updateVers();
      // Rekursiver Prozeduraufruf
7      self.korrigiere();
8    end
9    else
10     // Vers ist nicht analysierbar, Analyse bricht ab
      self.istAnalysierbar ← false;
11    end
12  end
13  else
14    // Speichere Ergebnis im Versmaß-Objekt
    self.speichereTransducerErgebnis();
    // Löse ggf. einen Zustandsübergang des DEA aus
15  end
16  return

```

ABBILDUNG 4.4: Algorithmus der Prozedur *korrigiere* für die Fehlerbehandlung.

Der Annotationsalgorithmus und die von ihm genutzten Klassen wurden mit Hilfe der Programmiersprache Python<sup>8</sup> implementiert.

## 4.4.2 Pakete und Klassen

### 4.4.2.1 Pakete

Die Annotations-Software ist auf die in Abbildung 4.5 dargestellten Pakete aufgeteilt. Die Datei *annotate.py* dient als Einstiegspunkt in das Programm und kann von der Nutzerin unter Angabe der Ein- und Ausgabedateien direkt aufgerufen werden. Die darüber hinaus für die Annotation benötigten Klassen sind nach funktionalen Gesichtspunkten auf weitere Pakete aufgeteilt.

Zunächst importiert *annotate.py* das Paket *preprocessing.py*, das die *preprocessor*-Klasse (Abschnitt 4.4.2.3) sowie eine Klasse *selector* enthält. Letztere dient lediglich der Zufallsselektion einer Teilmenge von Versen und ist nicht Teil des eigentlichen

<sup>8</sup><https://www.python.org/>.



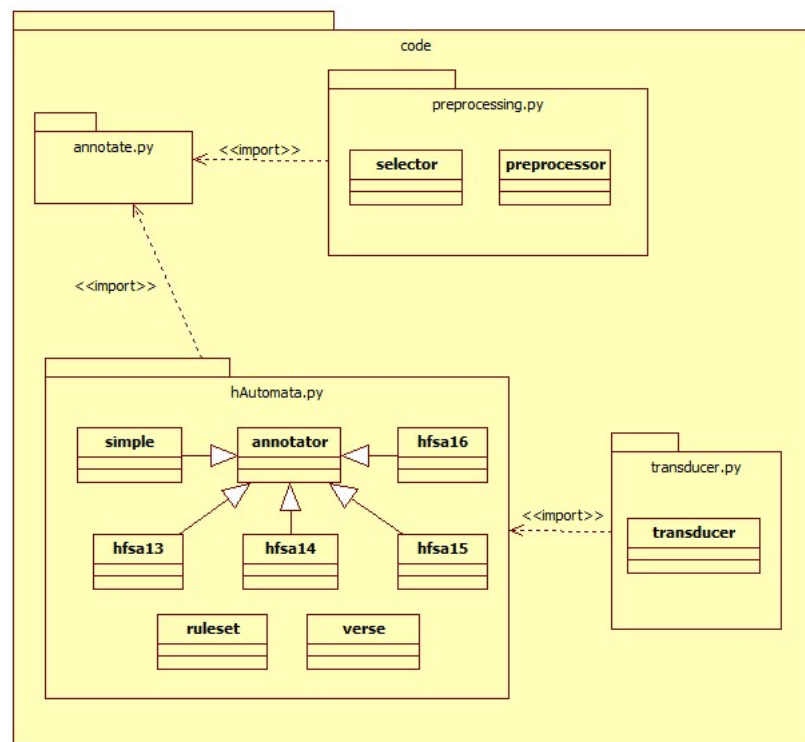


ABBILDUNG 4.5: UML-Paketdiagramm: Komponenten des Annotationsprogramms und Beziehungen zwischen ihnen.

Annotationsalgorithmus. Das ebenfalls von *annotate.py* importierte Paket *hAutomata.py* enthält alle Klassen, die für die regelbasierte Versmaßannotation benötigt werden: die DEAs *hfsa13*, *hfsa14*, *hfsa15*, *hfsa16* und *simple* (vgl. Abschnitt 4.4.2.7), die Klasse *ruleset* (Abschnitt 4.4.2.4), welche die für die Erkennung langer Silben benötigten linguistischen Regeln implementiert, und die Klasse *verse* (Abschnitt 4.4.2.5), welche Informationen zum aktuell bearbeiteten Vers kapselt. Darüber hinaus fasst die Elternklasse *annotator* (Abschnitt 4.4.2.6) gemeinsame Funktionalitäten der DEA-Klassen zusammen. Das Paket *transducer.py* wird von *hAutomata.py* importiert und enthält die Definition des Finite-State Transducers für die Vervollständigung teilannotierter Verse (Abschnitt 4.4.2.8). Darüber hinaus wurden außerhalb der genannten Pakete kurze Evaluationsskripte (vgl. Abschnitt 3.2) sowie ein Skript für den Export der Versdaten aus der von Kruse (2014) erstellten relationalen Datenbank implementiert.

#### 4.4.2.2 Klassengeflecht

Das für die regelbasierte Versmaßannotation benötigte Klassengeflecht ist in Abbildung 4.6 dargestellt. Die Darstellung enthält die Klassen der Pakete *hAutomata.py* und *transducer.py*. Zentral ist die Klasse *annotator*, von der die dedizierten DEA-Klassen erben. *annotator*-Instanzen nutzen Instanzen der Klassen *ruleset*, *verse* und *transducer*, definieren darüber hinaus den DEA-Klassen gemeinsame Attribute und kapseln Methoden, die von allen DEA-Klassen benötigt werden. Fallspezifische Unterscheidungen sind dagegen in den einzelnen DEA-Klassen implementiert.



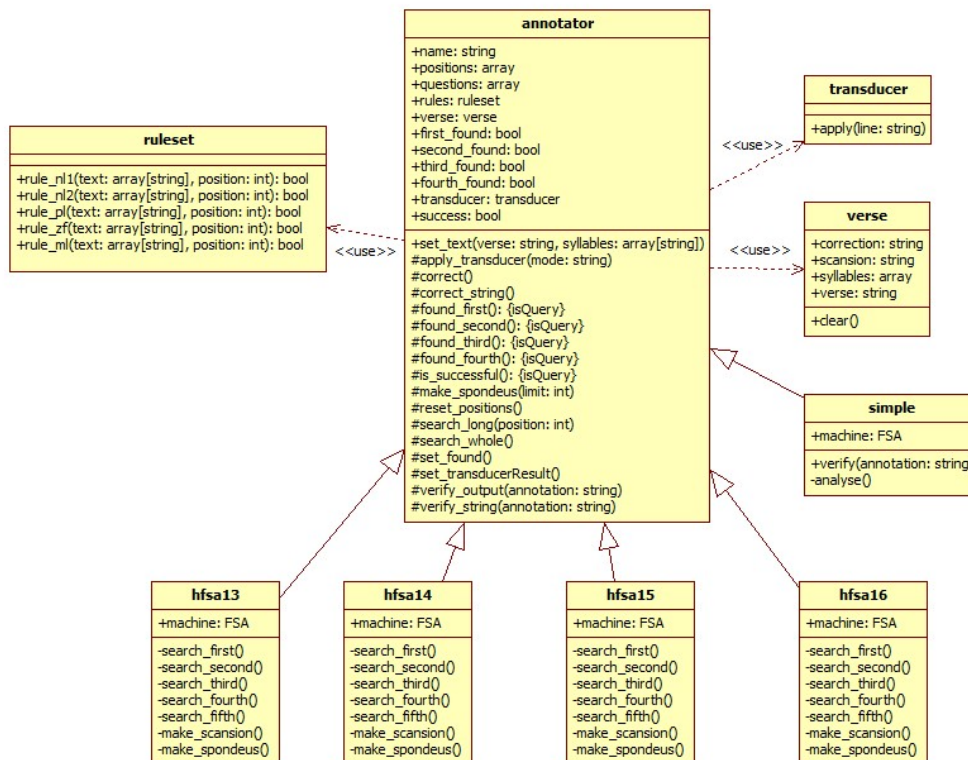


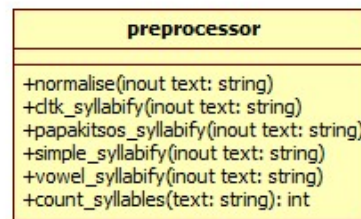
ABBILDUNG 4.6: UML-Klassendiagramm: *annotator*-Klasse und assoziierte Klassen.

#### 4.4.2.3 Klasse *preprocessor*

Ein UML-Klassendiagramm der *preprocessor*-Klasse findet sich in Abbildung 4.7. Die Klasse hat im Wesentlichen zwei Aufgaben:

- **Normalisierung des zu analysierenden Verses:** Unter Normalisierung ist hier das Entfernen aller Diakritika im Vers (außer Trema und Zirkumflex, vgl. Abschnitte 2.2 und 2.3.3) sowie die Umwandlung von Groß- in Kleinbuchstaben zu verstehen. Zudem werden Interpunktionszeichen entfernt. Dieser Schritt dient der Vereinfachung und Verkleinerung des zu verarbeitenden Zeichenbestandes und wird durch die Funktion *normalise* implementiert. Im Pseudo-Code in Abbildung 4.3 entspricht der Funktion der Aufruf *preprocessor.vereinfache()*.
- **Silbentrennung:** Es wurden mehrere Funktionen für die Zerlegung von Versen in Silben implementiert, da nicht ohne Weiteres ersichtlich war, welche Vorgehensweise sich als erfolgreich erweisen würde. Die Funktion *simple\_syllabify* nutzt für die Silbentrennung eine existierende Python-Bibliothek<sup>9</sup>. Die Funktion *vowel\_syllabify* implementiert keine Silbentrennung im engeren Sinne, sondern führt in Anlehnung an das Pascal-Skript von Höflmeier (1982, vgl. Abschnitt 3.6) eine Trennung nach jedem Vokal durch. Die Funktion *cltk\_syllabify* versucht eine Re-Implementierung des Silbentrennungs-Algorithmus aus der Python-Bibliothek CLTK (vgl. ebenfalls Abschnitt 3.6). Die Funktion *papakitosos\_syllabify* schließlich ist die umfangreichste Eigenimplementierung. Der Trennungsalgorithmus nutzt die von Papakitosos (2011)

<sup>9</sup><https://github.com/jtauber/greek-accentuation>.

ABBILDUNG 4.7: *preprocessor*-Klasse.

ausführlich dargestellten Silbentrennungsregeln, die im Rahmen linguistischer Konsultationen mit Mitarbeitern des Lehrstuhls von Prof. Blößner durch weitere Regeln ergänzt wurden. Bei der Trennung wird der Vers zeichenweise von links nach rechts durchlaufen, wobei das Vorliegen von Trennungsbedingungen mit Hilfe einer (noch recht übersichtlichen) *if-else*-Kaskade überprüft wird. Reguläre Ausdrücke werden genutzt, um komplexere lokale Phänomene wie beispielsweise die Trennung nach Elision zu behandeln. Im Pseudo-Code in Abbildung 4.3 entspricht der Silbentrennung der Aufruf *preprocessor.trenne()*.

Nicht zuletzt enthält die *preprocessor*-Klasse auch die Funktion *count\_syllables*, welche die Anzahl der Silben in einem bereits getrennten Vers zählt.

#### 4.4.2.4 Klasse *ruleset*

Die Klasse *ruleset* enthält die linguistischen Regeln für die Erkennung langer Silben, eine Instanz der Klasse steht jedem dedizierten DEA als Attribut zur Verfügung. Wie aus Abbildung 4.6 ersichtlich ist, wurden insgesamt fünf Funktionen implementiert, mit deren Hilfe die Regeln aus Abschnitt 2.3.3 auf Boolesche Ausdrücke abgebildet werden: Konkret entsprechen die Namen der implementierten Funktionen den Namen der in Abschnitt 2.3.3 in natürlicher Sprache formulierten Regeln. So implementiert beispielsweise die Funktion *rule\_nl1* die Regel  $R_{nl1}$ . Des Weiteren wurde eine Ausnahmeregel zur Erkennung von *muta cum liquida* implementiert. Jeder Funktion werden zur Prüfung der in Silben zerlegte Vers als String-Array sowie die zu prüfende Versstelle bzw. Silbe übergeben. Die Prüfung mittels regulärer Ausdrücke ergibt dann das Ergebnis *true*, wenn die Prüfbedingung vorliegt. Zur Veranschaulichung des Vorgehens illustriert Abbildung 4.8 die Implementierung der Regel zur Erkennung von *muta cum liquida* aus Abschnitt 2.3.3.

```

#muta cum liquida
def rule_m1(self, text, position):
    current = text[position-1]
    following = text[position]
    if re.search(r'[αιουεωη][βγδπκφχθ][λρνμ]', current+following):
        return True
  
```

ABBILDUNG 4.8: Beispiel für Implementierung linguistischer Regeln: Erkennung von *muta cum liquida*.

#### 4.4.2.5 Klasse *verse*

Eine Instanz der Klasse *verse* steht jedem DEA zur Verfügung. Sie enthält die zu einem gegebenen Zeitpunkt verfügbaren Informationen zum aktuell bearbeiteten Vers:

- den Verstext, d. h. das Ergebnis der Vorverarbeitung mittels *preprocessor*,
- die Silben, d. h. das Ergebnis der Silbentrennung,
- das vollständig oder teilweise erkannte Hexameterschema sowie eine möglicherweise während der Fehlerbehandlung erzeugte Variante des Schemas.

Beim Zurücksetzen des DEA vor Bearbeitung eines neuen Verses (vgl. Abschnitt 4.4.1) wird die Prozedur *clear* der Vers-Instanz aufgerufen, welche alle Vers-Attribute löscht, so dass Verwechslungen des aktuellen Verses mit zuvor analysierten Versen ausgeschlossen sind.

#### 4.4.2.6 Klasse *annotator*

Die Klasse *annotator* vererbt allen für die Versmaßannotation genutzten DEAs gemeinsame Attribute, Funktionen und Prozeduren. Zu den gemeinsamen Attributen gehören dabei neben dem Instanznamen sowie Instanzen der Klassen *ruleset*, *verse* und *transducer* auch vier boolesche Attribute, mit deren Hilfe gespeichert werden kann, wie viele Spondeen bereits im analysierten Vers gefunden wurden (maximal vier: Attribute *first\_found*, *second\_found*, *third\_found* und *fourth\_found*). Mit Hilfe des Integer-Arrays *positions* kann zudem vermerkt werden, welche Silben sicher als lang erkannt wurden. Das Attribut *questions* wird dagegen nur an wenigen Stellen dazu verwendet, Informationen über nicht eindeutig bewertbare Verse zu speichern. Nicht zuletzt kann mit Hilfe des booleschen Attributes *success* im Zuge der Plausibilitätsprüfung vermerkt werden, ob das gefundene Hexameterschema den linguistischen Regeln entspricht oder nicht.

Die einzige von außerhalb des Pakets *hAutomata.py* aufgerufene Prozedur der *annotator*-Klasse ist *set\_text*. Diese Prozedur wird aufgerufen, um der *verse*-Instanz des DEA aktuelle Versinformationen zuzuweisen (Zeilen 10 und 11 in Abbildung 4.3). Alle anderen Funktionen und Prozeduren werden von den DEAs während der Versanalyse oder bei Zustandsübergängen selbst aufgerufen:

- Die Prozedur *apply\_transducer* veranlasst die globale Versmaßanalyse mit Hilfe des Finite-State Transducers.
- Die Prozedur *correct* implementiert den in Abbildung 4.4 dargestellten Algorithmus für die Fehlerbehandlung.
- Die Prozedur *correct\_string* kapselt die während der Fehlerbehandlung benötigte buchstabenweise Analyse von Längen und Kürzen (Zeile 2 in Abbildung 4.4).
- Die vier in Abbildung 4.6 mit *isQuery* annotierten *found*-Funktionen geben die Werte der booleschen *found*-Attribute zurück. Diese sind für die Zustandsübergänge der DEAs relevant, da sie die lokale Suche steuern (vgl. Abschnitt 4.4.2.7).

```

def _search_long(self, position):
    if self.rules.rule_zf(self.verse.syllables, position):
        return True
    elif self.rules.rule_n11(self.verse.syllables, position):
        return True
    elif self.rules.rule_n12(self.verse.syllables, position):
        return True
    elif self.rules.rule_pl(self.verse.syllables, position) and not self.rules.rule_ml(self.verse.syllables, position):
        return True
    else:
        return False

```

ABBILDUNG 4.9: Programm-Code der Funktion *search\_long* aus der *annotator*-Klasse.

- Die Funktion *isSuccessful* gibt den Wert des Attributes *success* zurück. Dieses ist für die Zustandsübergänge der DEAs während der Überprüfung und Korrektur des Versmaßes von Bedeutung.
- Die Prozedur *make\_spondeus* speichert einen teilannotierten Vers im *scansion*-Attribut des lokalen *verse*-Objekts. Diese Teilannotation dient während der globalen Analyse als Eingabe des Finite-State Transducers.
- Die Prozedur *reset\_positions* setzt den DEA beim Übergang in den Wartezustand zurück (Zeile 9 in Abbildung 4.3), Informationen aus dem *positions*- und dem *questions*-Array sowie den booleschen *found*-Attributen werden gelöscht. *verse.clear()* wird aufgerufen.
- Die Prozedur *set\_found* setzt das entsprechende boolesche *found*-Attribut auf *true*, sobald während der Spondeensuche ein Spondeus sicher identifiziert wurde.
- Die Prozedur *set\_transducerResult* wählt – sofern der FST mehrere mögliche Lösungen ausgegeben hat –, das Ergebnis mit dem größten Gesamtgewicht (vgl. Abschnitt 4.4.2.8) und speichert es in der lokalen Vers-Instanz.
- Die Funktion *verify\_output* veranlasst die Plausibilitätsprüfung des erzeugten Hexameterschemas durch Aufruf der Funktion *verify\_string* und speichert deren Ergebnis im Attribut *success*. Alsdann wird ein Zustandsübergang des DEAs ausgelöst.

**Längenbestimmung** Etwas ausführlicher sollen die beiden verbleibenden Funktionen der *annotator*-Klasse erläutert werden, da sie für das Funktionieren des Annotationsalgorithmus von wesentlicher Bedeutung sind. Abbildung 4.9 präsentiert den Code der Funktion *search\_long*. Die Abbildung zeigt, dass in dieser Funktion die Regeln der Klasse *ruleset* (u. a. die Regel zu *muta cum liquida* aus Abbildung 4.8) direkt auf die Silben des Verses angewandt werden. Die implementierten Regeln lassen sich als prädikatenlogische Formeln – unter Nutzung der in Abschnitt 2.3.3 definierten Regelnamen – entsprechend den Gleichungen 4.3–4.6 ausdrücken:

$$R_{zf}(X) \Rightarrow \text{istLang}(X) \quad (4.3)$$

$$R_{n11}(X) \Rightarrow \text{istLang}(X)^{10} \quad (4.4)$$

<sup>10</sup>Die Regel ist so implementiert, dass das Vorliegen eines Hiats ausgeschlossen wird.

$$R_{nl2}(X) \Rightarrow \text{istLang}(X)^{11} \quad (4.5)$$

$$R_{pl}(X) \wedge \neg R_{ml}(X) \Rightarrow \text{istLang}(X) \quad (4.6)$$

Durch disjunktive Verknüpfung der Formeln erhält man die in Gleichung 4.7 dargestellte Formel. In allen Fällen, in denen die Bedingungen aus Gleichung 4.7 nicht erfüllt sind, gilt die Silbe als nicht eindeutig bestimmbar. Ein Vergleich der Formel aus Gleichung 4.7 mit den linguistischen Regeln aus Abschnitt 2.3.3 zeigt, dass Letztere durch die Funktion *search\_long* genau abgebildet werden.

$$R_{zf}(X) \vee R_{nl1}(X) \vee R_{nl2}(X) \vee (R_{pl}(X) \wedge \neg R_{ml}(X)) \Rightarrow \text{istLang}(X) \quad (4.7)$$

**Globale Analyse** Abbildung 4.10 zeigt den Programm-Code der Prozedur *search\_whole*. Diese Funktion wird von den DEAs im Zustand *Globale Analyse* aufgerufen und erhält einen teilannotierten Vers als Eingabe (sie entspricht also dem Aufruf in Zeile 37 von Abbildung 4.3). Im Vers sind dabei bekannte lange Silben durch das Zeichen „-“ markiert. Silben, deren Status vorerst noch unbekannt ist, tragen dagegen ein „?“ als Markierung. Die Abbildung zeigt, dass auf diese Silben zunächst die Prozedur *search\_long* angewandt wird, um mögliche weitere Längen zu identifizieren. Dieser Schritt ist nicht redundant, da die betreffenden Silben im Rahmen der bisherigen Analyse noch nicht betrachtet wurden. Das in Teilen bestimmte Versmaß wird dann durch Aufruf der entsprechenden Prozeduren dem Finite-State-Transducer übergeben. Falls dieser das Eingabewort akzeptiert, wird das Ergebnis gespeichert. Andernfalls wird das *success*-Attribut auf *false* gesetzt. Der Aufruf am Ende teilt dem DEA das Ende der Prozedur mit und löst einen Zustandsübergang aus.

#### 4.4.2.7 Hierarchische endliche Automaten

Die endlichen Automaten steuern die regelbasierte Versmaßannotation. Es wurden endliche Automaten für fünf unterschiedliche Versklassen implementiert:

- *hfsa13*: Für Verse mit 13 Silben. Dieser DEA versucht, vier Spondeen zu finden (vgl. Abbildung 2.2).
- *hfsa14*: Für Verse mit 14 Silben. Dieser DEA versucht, drei Spondeen zu finden.
- *hfsa15*: Für Verse mit 15 Silben. Dieser DEA versucht, zwei Spondeen zu finden.
- *hfsa16*: Für Verse mit 16 Silben. Dieser DEA versucht, einen Spondeus zu finden.
- *simple*: Für alle anderen Verse (mit mindestens 9 Silben). Dieser DEA führt lediglich die buchstabenweise Bestimmung von Längen und Kürzen und ggf. die Vervollständigung des Verses mittels FST durch.

Die Automaten wurden mit Hilfe einer existierenden Python-Bibliothek<sup>12</sup> implementiert. Sie steuern die Suche nach langen Silben (und bei Bedarf den Übergang in die globale Analyse), wobei jedem zu suchenden Spondeus ein eigener Zustand entspricht. Die Automaten sind zusätzlich als hierarchische Automaten konzipiert, d. h.

<sup>11</sup> Auch hier schließt die Implementierung der Regel das Vorliegen eines Hiats aus.

<sup>12</sup> <https://github.com/pytransitions/transitions>.

```

#function used to search the whole verse for long syllables (in fallback)
def _search_whole(self):
    s_units = list(self.verse.scansion)
    for x in range(1, len(s_units)-2): #no need to scan last two syllabs
        if s_units[x] == '?':
            search_result = self._search_long(x+1)
            if search_result:
                s_units[x] = '-'
    self.verse.scansion = ''.join(s_units)
    #apply finite-state transducer
    results = self._apply_transducer()
    #failure if transducer does not accept input string
    if len(results.items()) == 0:
        self.success = False
        self.scansionLength = 0
    else:
        lengths = [len(v) for v in results.values()]
        self.scansionLength = lengths[0]
        self._set_transducerResult(results, mode='fallback')
    self.verified()

```

ABBILDUNG 4.10: Programm-Code der Prozedur *search\_whole* aus der *annotator*-Klasse.

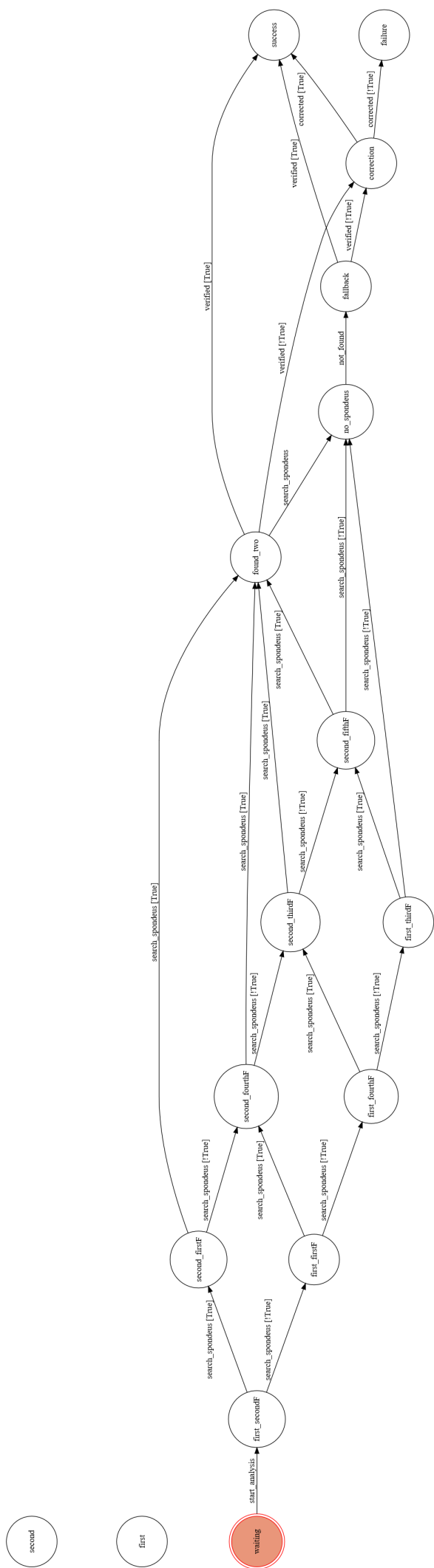
jeder Zustand hat eine Reihe von Unterzuständen: Entsprechend den Ausführungen von Papakitsos (2011) wird ein Spondeus zunächst im zweiten, dann im ersten Fuß usw. gesucht. Diese Suchläufe in bestimmten Versfüßen wurden als Unterzustände der Spondeensuche modelliert.

Abbildung 4.11 zeigt exemplarisch das Zustandsdiagramm für den DEA *hfsa15*. Die Abbildung wurde mit Hilfe der Bibliothek PyGraphviz<sup>13</sup> aus dem Programm-Code erstellt und weicht in Details von den Darstellungskonventionen für DEAs ab. Die beiden Elternzustände sind als nicht verbundene Kreise am linken Rand der Grafik visualisiert, während die Zustandsübergänge der Grafik stets einen Kindzustand als Start und Ziel haben. Die Namen der Kindzustände sind an die Namen der Elternzustände angehängt, beispielsweise hat der Zustand *first* (in dem der erste Spondeus gesucht wird) den Kindzustand *first\_firstF*, der die Suche nach dem ersten Spondeus im ersten Versfuß implementiert. Der Startzustand *waiting* ist rot markiert. Nicht markiert ist dagegen der akzeptierende Zustand *success*. Der Zustand *failure* ist der Misserfolgzustand, der das Scheitern der Analyse anzeigt. Nicht zuletzt entspricht der Zustand *fallback* in der Grafik dem logischen Zustand *Globale Analyse*.

Die Grafik zeigt, dass die Spondeensuche durch den endlichen Automaten vollständig modelliert wird. In jedem (Kind-)Zustand wird die dem interessierenden Versfuß entsprechende Suchoperation ausgeführt. Ein Beispiel für eine solche Operation ist die Operation Prozedur *search\_fourth*, die in Abbildung 4.12 für den DEA *hfsa13* exemplarisch dargestellt ist: Hier wird im vierten Versfuß nach einem Spondeus gesucht.

<sup>13</sup><http://pygraphviz.github.io/>.





HFSA15

ABBILDUNG 4.11: Zustandsdiagramm für den DEA hfsa15.



Da die Gesamtstruktur des Verses zum Zeitpunkt der Spondeensuche unbekannt ist, muss die Längenprüfung auf mehreren (Kandidaten-)Silben ausgeführt werden (vgl. hierzu auch die Ausführungen von Papakitsos, 2011). In Abbildung 4.12 sind dies die Silben sieben, acht und neun, die als Eingabeparameter der Funktion *search\_long* der *annotator*-Klasse übergeben werden. Nur zwei konsekutive Längen werden als Spondeus akzeptiert und entsprechend im *positions*-Array der DEA-Instanz vermerkt. Ist ein Spondeus gefunden, wird auch die Prozedur *set\_found* aufgerufen, die das dem gesuchten Spondeus entsprechende *found*-Attribut auf „true“ setzt (vgl. den vorhergehenden Abschnitt). Allerdings werden auch einzelne identifizierte Längen im *positions*-Array vermerkt. Der Aufruf der Prozedur *search\_spondeus* löst am Ende der Funktion den Übergang in einen neuen Zustand aus, da die Suche für den gegebenen Versfuß dann abgeschlossen ist. Wie Abbildung 4.11 zeigt, ist für die Auswahl des Folgezustands der Wert des dem Elternzustand entsprechenden booleschen *found*-Attributs maßgeblich.

Darüber hinaus implementieren die DEA-Klassen, wie in Abbildung 4.6 zu sehen, noch die Prozeduren *make\_spondeus* und *make\_scansion*. Letztere erzeugt aus dem *positions*-Array das auszugebende Versmaß. Die Prozedur *make\_spondeus* ruft dagegen die gleichnamige Prozedur der Elternklasse auf, die dann einen teilannotierten Vers für die Vervollständigung durch den FST im Zustand *Globale Analyse* speichert.

#### 4.4.2.8 Klasse *transducer*

Abbildung 4.13 zeigt eine **verkürzte** Darstellung des durch die Klasse *transducer* implementierten Finite-State Transducers, welche die Funktionsweise des FSTs verdeutlichen soll. Der im Annotationsprogramm implementierte Transducer enthält zwischem dem ersten und dem letzten Versfuß weitere Zustände und Übergänge, die allerdings analog zu den in der Abbildung dargestellten Zuständen  $q_3$ ,  $q_4$  und  $q_{33}$  sowie den entsprechenden Übergängen definiert wurden. Der Finite-State Transducer wurde mit Hilfe einer dedizierten Python-API für die *Helsinki Finite-State Transducer Technology Tool-Suite*<sup>14</sup> entwickelt.

Der Abbildung ist zunächst zu entnehmen, dass es sich um einen nicht-deterministischen, gewichteten FST handelt. Der Transducer akzeptiert sowohl vollständige als auch teilannotierte, wohlgeformte Hexameter-Verse. Bekannte Längenzuweisungen werden bei der Verarbeitung nicht verändert, lediglich die Fragezeichen, die längenmäßig nicht bestimmte Silben markieren, werden entsprechend den Wohlgeformtheitsregeln für Hexameter-Verse durch Symbole des Ausgabealphabets ersetzt. Beispielsweise zeigt Abbildung 4.13 einen Übergang vom Zustand  $q_0$  in den Zustand  $q_1$  unter Ersetzung des Fragezeichens durch einen Bindestrich. Diese Regel bezieht sich auf die erste Silbe des ersten Versfußes, die stets lang sein muss<sup>15</sup>. Deutlich wird außerdem, dass die Anzahl der Zustände des FSTs der Anzahl der in einem Hexameter-Vers maximal möglichen Silben (plus ein Startzustand  $q_0$ ) entspricht, d. h. für jeden Versfuß stehen immer zwei Zustände zur Verfügung. Zusätzlich existieren im Sinne des Nicht-Determinismus optionale Zustände  $q_{11}$ ,  $q_{33}$  etc. für jeden Versfuß (bis auf den letzten, vgl. Abbildung 4.13) die bei Vorliegen eines

<sup>14</sup><https://hfst.github.io/python/3.12.1/index.html>.

<sup>15</sup>Die Regel kann nur im Rahmen einer Fehlerbehandlung zum Einsatz kommen, wenn bei der buchstabenweisen Längenbestimmung der erste Vokal nicht eindeutig bestimmt werden konnte. Während der im Allgemeinen zuvor durchgeführten lokalen Suche wird die erste Silbe dagegen gar nicht analysiert, da sie ohnehin als lang gilt.

```

def _search_fourth(self):
    seventh = self._search_long(7)
    eighth = self._search_long(8)
    ninth = self._search_long(9)
    if seventh and eighth:
        self.positions.append(7)
        self.positions.append(8)
        self._set_found()
    elif eighth and ninth:
        self.positions.append(8)
        self.positions.append(9)
        self._set_found()
    else:
        if seventh:
            self.positions.append(7)
        if eighth:
            self.positions.append(8)
        if ninth:
            self.positions.append(9)
    self.search_spondeus()

```

ABBILDUNG 4.12: Programm-Code der Prozedur *search\_fourth* des DEA *hfsa13*.

passenden Eingabeworts durchlaufen werden können, es aber nicht müssen<sup>16</sup>. Diese Zustände modellieren das Auftreten möglicher Spondeen im Vers.

Die Verteilung möglicher Daktylen im Vers wird nun mit Hilfe von Gewichten an den Zustandsübergängen modelliert, die in Abbildung 4.13 durch das jeweils dritte Label an den Zustandsübergängen angegeben werden. Hierbei folgt die Implementierung den Ausführungen von Papakitsos (2011), demzufolge ein Daktylus beispielsweise im zweiten Versfuß häufiger auftritt als im ersten<sup>17</sup>. Die Gewichte an den Zustandsübergängen bewirken bei Vorliegen mehrerer Möglichkeiten der Verteilung von Daktylen in einem teilannotierten Vers unterschiedliche Gesamtgewichte der möglichen Lösungen und unterstützen mithin die Auflösung komplexer, da stark unterbestimmter Verse. Existiert dagegen nur eine Lösungsmöglichkeit, sind die Gewichte irrelevant.

Die Analyse eines teilannotierten Verses durch den Finite-State Transducer kann zu unterschiedlichen Ergebnissen führen:

- (a) Der Transducer akzeptiert den eingegebenen, teilannotierten Vers.
  - (i) Der Transducer erzeugt mehrere gültige Ausgabewörter.
  - (ii) Der Transducer erzeugt genau ein Ausgabewort.

<sup>16</sup>Um diesen Umstand zu verdeutlichen, wurden die Zustandsnamen durch Verdopplung der Indizes der Vorgängerzustände gebildet.

<sup>17</sup>Bei Vorliegen eines annotierten Korpus wäre es möglich gewesen, besser angepasste Gewichtungen aus den Daten zu entnehmen, vgl. die in Abschnitt 3.5 kurz vorgestellte Arbeit von Greene, Bodrumlu und Knight (2010).

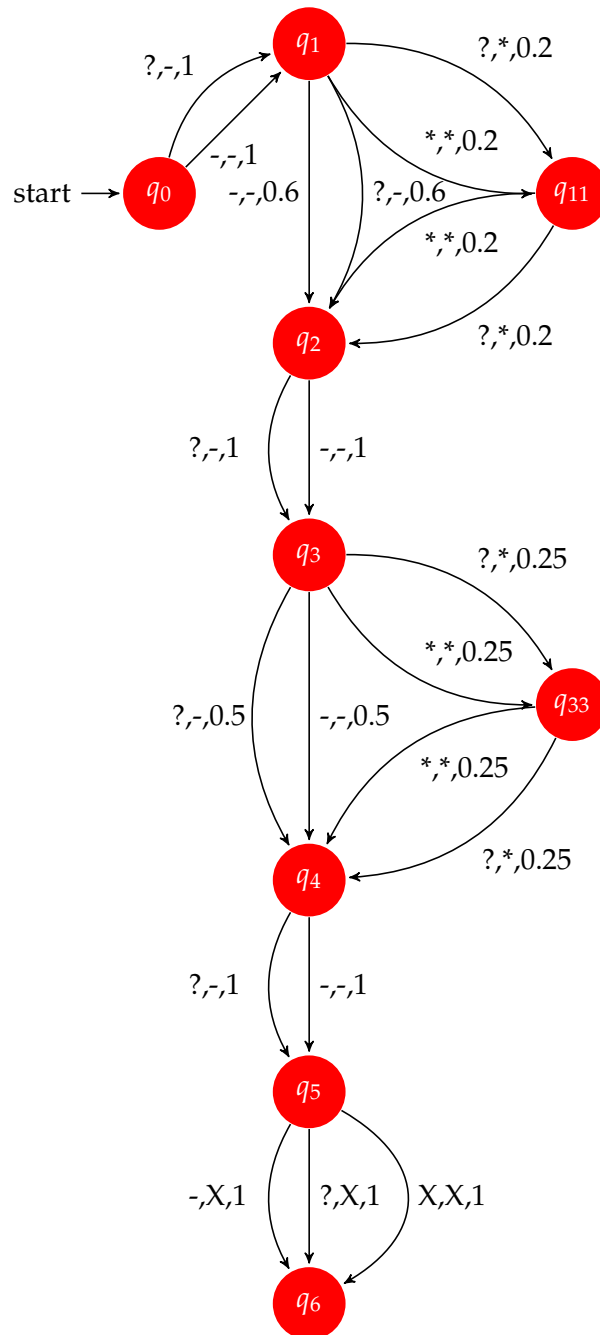


ABBILDUNG 4.13: Gewichteter Finite-State Transducer für die Versmaßvervollständigung (verkürzte Darstellung).

- (b) Der Transducer verwirft den eingegebenen Vers. Es wird kein Ausgabewort erzeugt.
  - (i) Das Eingabewort war fehlerhaft vorannotiert.
  - (ii) Der Vers ist kein gültiger Hexameter.

In allen Fällen wird das Transducer-Ergebnis durch die hierarchischen endlichen Automaten weiterverarbeitet wie in Abschnitt 4.4.2.6 beschrieben. Beispielsweise wird bei Vorliegen mehrerer Ausgabewörter das Ergebnis mit dem höchsten Gesamtgewicht ausgewählt. Wird dagegen auch im Ergebnis der Fehlerbehandlung kein Ausgabewort erzeugt, gilt der Vers als nicht analysierbar und die Verarbeitung bricht ab.

Zusätzlich zur Definition des Finite-State Transducers enthält die Klasse *transducer* auch eine Funktion *apply*, mit deren Hilfe der Finite-State Transducer wie in Abschnitt 6 beschrieben auf ein Eingabewort angewandt werden kann. Diese Funktion wird während der Analyse durch die Prozedur *apply\_transducer* der *annotator*-Klasse aufgerufen (vgl. Abschnitt 4.4.2.6).

## 4.5 Zusammenfassung

In diesem Kapitel wurde die implementierte Annotations-Software im Hinblick sowohl auf die zugrunde liegenden theoretischen Konzepte als auch auf den gewählten Annotationsalgorithmus und die praktische Strukturierung des Codes erläutert. Im Ergebnis der Implementierungsarbeiten ist eine modular aufgebaute und damit auch modular nutzbare sowie jederzeit erweiterbare Annotations-Software entstanden. Beispielsweise kann das implementierte Vorverarbeitungsmodul – insbesondere die Funktion zur Silbentrennung – auch unabhängig von einer möglichen Hexameter-Annotation genutzt werden.

Im Unterschied zur Implementierung beispielsweise von Höflmeier (1982) handelt es sich bei der hier vorgestellten Software zudem um ein Programm, welches das Problem der Hexameter-Annotation vollautomatisch bearbeitet und die Auflösung komplexer Verse nicht der Nutzerin überlässt<sup>18</sup>. Ein weiterer Vorteil im Vergleich zu den häufig sehr stark auf die Anwendung feingranularer linguistischer Regeln fixierter Implementierungen durch Altphilologen (vgl. Abschnitt 3.6) besteht darin, dass es gelungen ist, den Annotationsprozess durch Anwendung wohldefinierter theoretischer und computerlinguistischer Konzepte – deterministische endliche Automaten und Finite-State Transducer – algorithmisch zu formalisieren. Hieraus ergeben sich u. a. die folgenden Vorteile:

1. Das Problem der Hexameter-Annotation wurde gängigen Methoden des Software Engineering und der objektorientierten Programmierung zugänglich gemacht. Dies ermöglicht eine weitere Optimierung des Annotationsalgorithmus z. B. im Hinblick auf die Nachvollziehbarkeit und Wartbarkeit des Programm-Codes oder auf möglichst effiziente Implementierungen.
2. Auch die genutzten theoretischen Konzepte (Automaten) bieten aufgrund ihrer wohldefinierten Eigenschaften Ansatzpunkte für weitere Optimierungen. Wünschenswert wäre beispielsweise eine Zusammenfassung der implementierten DEAs durch Parametrisierung oder Verknüpfung.

---

<sup>18</sup>Freilich muss die im nächsten Kapitel noch durchzuführende Evaluation zeigen, welche Qualität die vollautomatisch berechneten Annotationsergebnisse haben.

3. Nicht zuletzt bieten die implementierten Automaten auch den Vorteil einer einfachen und intuitiven Dokumentierbarkeit: Der in Abbildung 4.11 dargestellte endliche Automat ist durch die grafische Form der Darstellung leicht zu erfassen und kann auch als Hilfsmittel beim Einüben von Strategien für die manuelle Hexameter-Annotation etwa in der Ausbildung von Altphilologen genutzt werden. Auf diese Weise trägt die gefundene Lösung der eingangs formulierten Anforderung einer auch für Altphilologen nachvollziehbaren Versannotation Rechnung.

Erwähnung verdient noch, dass sich der für die Versmaßannotation genutzte Satz linguistischer Regeln als sehr klein erwiesen hat. In der hier dargestellten, auf Konsultationen mit Prof. Blößner und Kollegen zurückgehenden Form bilden die Regeln diejenigen Indizien ab, auf die geschulte Experten (Altphilologen) ihre eigenen Annotationsentscheidungen stützen, d. h. die Regeln beschreiben ein klassisch-systemlinguistisches Verständnis von Sprache, nach dem ein linguistisches Phänomen mittels einer endlichen Menge binärer (liegt vor/liegt nicht vor) sprachlicher Merkmale beschrieben werden kann. Weder komplexe Interaktionen zwischen den gewählten Merkmalen (die linguistischen Regeln konnten durch einfache logische Operationen miteinander verknüpft werden, vgl. Abschnitt 4.4.2.6) noch über den unmittelbaren lokalen Kontext hinausgehende Einflüsse (wie etwa bei der Annotation russischer Wortbetonungen im Kontext, vgl. Abschnitt 3.3) mussten beachtet werden, so dass der linguistische Regelsatz umstandslos mit Hilfe von regulären Ausdrücken implementiert werden konnte. Sobald für die weitere Forschungsarbeit umfangreichere, zuverlässig annotierte Daten vorliegen, sind im Hinblick auf die Nutzung linguistischen Expertenwissens interessante Erweiterungen denkbar, beispielsweise durch das Erlernen optimaler Regelmengen etwa mit genetischen Algorithmen oder durch die statistische Analyse unterschiedlichster linguistischer Merkmale. Solche Studien könnten gerade auch im Hinblick auf aktuell noch nicht diskret modellierbare Phänomene wie Krasis oder Synizese neue Erkenntnisse liefern.

## Kapitel 5

# Evaluation

Nah ist  
 Und schwer zu fassen der Gott.  
 Wo aber Gefahr ist, wächst  
 Das Rettende auch.  
 Im Finstern wohnen  
 Die Adler und furchtlos gehn  
 Die Söhne der Alpen über den Abgrund weg  
 Auf leichtgebaueten Brücken.  
 Drum, da gehäuft sind rings  
 Die Gipfel der Zeit, und die Liebsten  
 Nah wohnen, ermattend auf  
 Getrenntesten Bergen,  
 So gib unschuldig Wasser,  
 O Fittige gib uns, treuesten Sinns  
 Hinüberzugehn und wiederzukehren.

Friedrich Hölderlin, aus dem Gedicht „Patmos“

### 5.1 Vorbemerkung

In diesem Kapitel wird die Evaluation der vorstehend beschriebenen Annotations-Software erläutert. Dazu wird in Abschnitt 5.2 zunächst dargestellt, auf welcher Datengrundlage, wie und gegen welche Baseline evaluiert wird. Alsdann werden in den Abschnitten 5.3 und 5.4 die bei der Evaluation der Silbentrennung und der eigentlichen Hexameter-Annotation ermittelten Performanzwerte genannt. Abschnitt 5.5 beschließt dieses Kapitel.

### 5.2 Evaluationsdaten, Metriken und Baseline

Zur Evaluation der Annotations-Software werden die in Abschnitt 3.2 genannten Evaluationsmetriken genutzt. Die Evaluation des Silbentrennungsalgorithmus kann dabei sowohl silben- als auch versweise erfolgen. Für die Evaluation der Hexameter-Annotation erscheint dagegen die versweise Evaluation als praktikabelste Variante.

Als Datengrundlage für die Evaluation dient eine Zufallsauswahl von Versen aus dem Gesamtkorpus, die mit Hilfe der *selector*-Klasse aus dem *preprocessing*-Paket der Annotations-Software (vgl. Abbildung 4.5) erzeugt wurde. Tabelle 5.1 gibt einen Überblick über diesen Datensatz: Insgesamt wurden 346 Verse selektiert, womit eine Abdeckung von etwa 1 % in Bezug auf das Gesamtkorpus erreicht wird. Die

	IL	OD	TH	ER	AS	HF	HY
Silbentrennung	159	12	–	–	–	–	–
Versmaß	159	122	11	8	5	18	23
<b>Gesamtkorpus</b>	<b>15.875</b>	<b>12.150</b>	<b>1066</b>	<b>840</b>	<b>487</b>	<b>1835</b>	<b>2344</b>

TABELLE 5.1: Überblick über die Evaluationsdatensätze. Für beide Datensätze ist die Verteilung der Verse über die größeren Subkorpora angegeben. Kürzel der Subkorpora entsprechend Tabelle 2.4.

Verteilung der Verse über die einzelnen Subkorpora entspricht in etwa der Verteilung im Gesamtkorpus, wobei jedoch nur die größeren Subkorpora (> 100 Verse) berücksichtigt wurden. Aus dem erstellten Evaluationsdatensatz wurden für die Evaluation der Silbentrennung die ersten 171 Verse von Mitarbeitern des Lehrstuhls von Prof. Blößner manuell in Silben zerlegt. Für die Evaluation der Hexameter-Annotation nahmen dieselben Mitarbeiter dagegen eine manuelle Annotation auf dem gesamten Textdatensatz vor.

Als Baseline für die Evaluation der Hexameterannotation wird die Python-Bibliothek von Hope Ranker (vgl. Abschnitt 3.6) genutzt, da es sich bei dieser nicht nur um eine relativ rezente, sondern auch – dies haben eine Analyse des einsehbaren Codes sowie erste Tests gezeigt – um eine saubere und leistungsfähige, vollautomatische Implementierung handelt.

### 5.3 Silbentrennung

Da die Qualität der Silbentrennung eine wesentliche Einflussgröße für den hier entwickelten Annotationsalgorithmus darstellt, scheint eine gesonderte Evaluation gerechtfertigt. Als Evaluationsmaß wird die Annotationsgenauigkeit, d. h. der Anteil der korrekt getrennten Silben, angegeben. Evaluiert wird die Performanz der Funktion *papakitsos\_syllabify* (vgl. Abschnitt 4.4.2.3). Ein Vergleich mit einer Baseline scheint nicht sinnvoll, da bereits während der Entwicklung deutlich wurde, dass die ebenfalls in das *preprocessing*-Paket integrierten möglichen Konkurrenten deutlich schlechtere Performanzwerte erreichen. Der folgende Kasten fasst das Evaluationsergebnis zusammen:



#### Evaluationsergebnisse Silbentrennung

Evaluierte Verse: 171  
 Evaluierte Silben: 2695  
 Versweise Genauigkeit: 0,82  
 Silbenweise Genauigkeit: 0,98

Der relativ große Performanzunterschied zwischen silben- und versweiser Evaluation deutet darauf hin, dass in vielen Fällen lediglich einzelne Silben einen Trennungsfehler aufweisen. Eine Fehleranalyse bestätigt dieses Ergebnis: Das Gros identifizierter Trennungsfehler ist auf die fehlerhafte Zuordnung von Konsonanten zwischen Vokalen (Zuordnung zur ersten vs. Zuordnung zur zweiten Silbe) insbesondere bei zusammengesetzten Wörtern zurückzuführen. Für eine Lösung dieses Problems wäre eine deutlich komplexere Implementierung unter Rekurs auf semantisches oder lexikalisches Wissen erforderlich gewesen. Da für die Hexameter-Annotation jedoch in erster Linie die Vokale von Bedeutung sind, kann davon ausgegangen werden,



	Genauigkeit	Abdeckung	F-Maß
Grundalgorithmus mit lokaler Suche	0,97	0,66	<b>0,78</b>
– mit globaler Analyse	0,94	0,97	<b>0,96</b>
Baseline	0,98	0,98	<b>0,98</b>
Kompletter Algorithmus	0,95	1,00	<b>0,98</b>

TABELLE 5.2: Evaluationsergebnisse Hexameterannotation.

dass die Trennungsfehler für das hier verfolgte Ziel unerheblich sind. Der implementierte einfache Silbentrennungsalgorithmus scheint mithin eine für die automatische Versmaßannotation ausreichende Qualität aufzuweisen.

## 5.4 Versmaß-Annotation

Tabelle 5.2 enthält die Ergebnisse für die Evaluation der Hexameterannotation. Es wurden verschiedene Nutzungsformen des Annotationsalgorithmus evaluiert (vgl. hierzu auch Abschnitt 4.4.1):

- die Grundform des Algorithmus, die lediglich die lokale Spondeensuche mit Hilfe der DEAs implementiert (Zeile 1),
- die erweiterte Form des Algorithmus, die zusätzlich zur lokalen Suche auch die globale Analyse mit Hilfe des Finite-State Transducers umsetzt (Zeile 2),
- die Vollform des Algorithmus einschließlich Fehlerbehandlung (Zeile 4).

Angegeben sind des Weiteren die Evaluationsergebnisse für die Baseline von Hope Ranker (Zeile 3). Zunächst fällt auf, dass bereits die Baseline – insbesondere im Vergleich zu den in Tabelle 3.2 zusammengetragenen Ergebnissen – ein hervorragendes Ergebnis vorgibt, das schwer zu übertreffen ist. Der in Anlehnung an Papakitsos (2011) implementierte Grundalgorithmus bleibt denn auch weit hinter dem Baseline-Ergebnis zurück, da zwar selbst diese einfache Suchlösung bereits eine recht hohe Präzision aufweist, aber unter einer schwachen Abdeckung leidet. Die Hinzunahme der globalen Analyse mit Hilfe eines FSTs bringt jedoch einen deutlichen Performanzsprung infolge einer starken Verbesserung der Abdeckung. Insbesondere wird auch der von Höflmeier (1982) genannte Benchmark in Höhe von 90 % maschinell analysierbarer Verse signifikant übertroffen<sup>1</sup>. Das in der letzten Zeile von Tabelle 5.2 angegebene Endergebnis bei Nutzung aller Komponenten (also insbesondere auch der Fehlerkorrektur gemäß dem Algorithmus aus Abbildung 4.4) bewegt sich im Bereich der Baseline.

## 5.5 Diskussion

Die Evaluation belegt die Konkurrenzfähigkeit des Annotationsalgorithmus. Bereits eine kurze Analyse zeigt, dass sich insbesondere Verse des Subkorpus HF (Hesiod-Fragmente, vgl. Tabelle 2.4) als maschinell nicht analysierbar erweisen: Häufig können bereits während der Silbentrennung nicht genügend Silben gefunden werden, d. h. die überlieferten Verse sind keine gültigen Hexmater und werden folgerichtig

<sup>1</sup>Anzumerken ist allerdings, dass die von Höflmeier genannte Größe kein Abdeckungsmaß angibt, sondern lediglich den Anteil derjenigen Verse, für die überhaupt eine Ausgabe erfolgt. Der ermittelte Unterschied ist also noch wesentlicher.

nicht als solche annotiert. Für nahezu alle anderen Verse im analysierten Datensatz wird ein Hexameterschema ausgegeben.

Verbesserungspotentiale ergeben sich insbesondere im Hinblick auf die Genauigkeit der Versmaßannotation. Diese kann sowohl durch eine verbesserte Formulierung der lokalen linguistischen Regeln – mögliche Ansätze hierzu wurden am Ende von Abschnitt 4.5 genannt – als auch durch die bislang nicht implementierte Behandlung von Krasis und Hiatkürzung erhöht werden. Unter der Bedingung, dass auf ausreichend große Trainingsdatensätze zugegriffen werden kann, erscheint die Modellierung der letztgenannten Phänomene mit Hilfe sequentieller Ansätze wie beispielsweise *Conditional Random Fields* oder Hidden-Markov-Modellen als besonders aussichtsreich. Andernfalls sind entsprechende Routinen aber auch in den in dieser Arbeit beschriebenen Algorithmus integrierbar.

## Kapitel 6

# Anwendung

### 6.1 Vorbemerkung

In diesem Abschnitt soll beschrieben werden, welche Ergebnisse bei der Anwendung des entwickelten Annotationsalgorithmus auf das gesamte Korpus homerischer Texte erzielt wurden. Insbesondere soll ein Vergleich der Annotationsergebnisse mit dem Output des Programms WinMetrix von Höflmeier (1982) Hinweise auf die Zuverlässigkeit der Gesamtannotation erbringen. Ihre Relevanz erhält diese Analyse insbesondere durch praktische Erfordernisse am Lehrstuhl von Prof. Blößner, wo WinMetrix bereits genutzt wird: Da eine manuelle Evaluation der ca. 35.000 Korpusverse mit vertretbarem Aufwand nicht möglich ist, soll eine Analyse der Übereinstimmungen Hinweise auf zuverlässig annotierte Verse geben.

Dafür wird zunächst in Abschnitt 6.2 beschrieben, in welcher Form und mit welchem Erfolg das Korpus annotiert wurde. Des Weiteren wird die Annotationsqualität des Programms WinMetrix von Höflmeier (1982) evaluiert (Abschnitt 6.3). Alsdann werden Übereinstimmungen zwischen den Annotationen der beiden Algorithmen analysiert (Abschnitt 6.4). Abschließend werden Kennzahlen für die Bewertung der Gesamtannotation mit Hilfe des neu entwickelten Algorithmus erhoben (Abschnitt 6.5). Wie üblich beschließt eine kurze Diskussion in Abschnitt 6.6 dieses Kapitel.

Anzumerken ist, dass die für den Vergleich der Annotationsergebnisse erforderliche Zusammenführung unterschiedlicher Datensätze – der hier genutzten Verse aus der SQL-Datenbank von Kruse (2014) sowie der am Lehrstuhl von Prof. Blößner genutzten abweichenden Korpusversion im Beta-Code-Format und schließlich des Outputs von WinMetrix – insofern mit einem geringen Datenverlust verbunden war, als nicht alle Verse ihren jeweiligen Pendants in den anderen Datensätzen zugeordnet werden konnten. Für den Vergleich der Annotationen werden daher „lediglich“ 34.599 Verse betrachtet.

### 6.2 Annotation des Gesamtkorpus

Der Annotationsalgorithmus ließ sich – erwartungsgemäß – auf das gesamte Korpus anwenden und erzeugte ein Versmaß für 33.380 der analysierten 34.599 Verse (96 %), wobei sich das Gros der nicht annotierten Verse auf das Subkorpus HF (vgl. Abschnitt 2.4) zu konzentrieren scheint, in dem viele Verse allein schon im Hinblick auf die Silbenzahl keine gültigen Hexameter sein können. Die Ausgabe der Ergebnisse erfolgt in der in Abbildung 6.1 dargestellten Form: Zu jedem Vers werden ein philologischer Index, der Verstext, das Ergebnis der Silbentrennung und das ermittelte Hexameterschema ausgegeben. Ein boolescher Wert zeigt zudem an, ob der Annotationsalgorithmus eine Synzese identifiziert hat. Die beiden Integer-Werte

am Zeilenende geben die Anzahl der während der lokalen und globalen Analyse sowie ggf. der Fehlerbehandlung gefundenen potentiellen Lösungen an. Für nicht annotierte Verse wird der Text „NOT RESOLVED“ ausgegeben.

```
IL.1,1 Μῆνιν ἄειδε θεὰ Πηληϊάδεω Ἀχιλῆος μῆνιν α.ει.δε θε.α πη.λη.ιτ.α.δε.ω α.χι.λη.ος
--*-----*-X True 1 0
IL.1,2 „ούλομένην, ἣ μυρί' Ἀχαιοῖς ἄλγε' ἔθηκε,“ ου.λο.με.νην η μυ.ρι α.χαι.οι.ζ αλ.γε
ε.θη.κε --* --* --* -X False 1 0
IL.1,3 „πολλὰς δ' ἰφθίμους ψυχὰς Ἀϊδὶ προΐαψεν“ πολ.λας δι.φθι.μους ψυ.χας α.ιτ.δι
προ.ιτ.α.ψεν -----*-X False 3 0
IL.1,4 ἥρώων, αὐτοὺς δὲ ἑλώρια τεῦχε κύνεσσιν η.ρω.ων αυ.τους δε ε.λω.ρι.α τευ.χε κυ.νε.σ.σιν --
--* --* --* -X False 1 0
IL.1,5 οἰωνοῖσι τε πᾶσι, Διὸς δ' ἐτελείετο βουλή, οι.ω.νοι.σι τε πα.σι δι.ος δε.τε.λει.ε.το
βου.λη --* --* --* -X False 1 0
```

ABBILDUNG 6.1: Ausschnitt aus Gesamtergebnis: erste fünf Verse der Ilias.

### 6.3 Qualität der Gesamtannotation mit WinMetrix

Zunächst soll die Qualität der WinMetrix-Annotation mit denselben Mitteln evaluiert werden wie zuvor in Abschnitt 5 die Qualität der eigenen Annotation. Das heißt, dass das Annotationsergebnis mit Hilfe derselben Metriken mit dem manuell erstellten Evaluationsdatensatz verglichen werden soll. Die Evaluation ergibt die folgenden Werte:

#### Evaluationsergebnisse WinMetrix

Evaluierte Verse: 346  
Genauigkeit: 0,97  
Abdeckung: 0,97  
F-Maß: 0,97

Damit liegen die Evaluationsergebnisse für WinMetrix minimal unter den in Tabelle 5.2 genannten Ergebnissen sowohl für die eigene Annotations-Software als auch für die Baseline. Eine Anmerkung ist hier allerdings angebracht: Im Unterschied zu den genannten Programmen arbeitet WinMetrix nicht vollautomatisch. Sülzmann (2018) zufolge berechnet WinMetrix für 3129 der vom Autor betrachteten 34.723 Verse<sup>1</sup> keine Ausgabe, sondern überlässt dem Nutzer die Eingabe des korrekten Hexmatterschemas. Dies entspricht etwa 9 % der untersuchten Versmenge. Wird dieser Anteil zusammen mit einem geschätzten Anteil von fehlerhaften Annotationen in Höhe von 3 % (1041 Verse, s. o.) für die Berechnung einer oberen Leistungsgrenze genutzt, ergeben sich die folgenden Werte:

#### Abschätzung einer oberen Leistungsgrenze für WinMetrix unter Berücksichtigung manueller Ergänzungen


Geschätzte Genauigkeit: 0,97  
Geschätzte Abdeckung:  $\text{Abdeckung} = \frac{34.723 - 3129 - 1041}{30.553 + 3129} = \frac{30.553}{33.682} = 0,91$   
F-Maß: 0,94

<sup>1</sup>Die Abweichung von den in Abschnitt 2.4 beschriebenen Ausgangsdaten ist auf die Verwendung einer anderen Korpusversion am Lehrstuhl von Prof. Blößner zurückzuführen.

Die ermittelte Differenz zu den in Tabelle 5.2 zusammengetragenen Ergebnissen trägt dem Unterschied zwischen teil- und vollautomatischer Annotation Rechnung. Eine Stärke von WinMetrix ist insbesondere die hohe Genauigkeit der Annotation, eine genauere Analyse des Programms könnte also Bestandteil einer Strategie für die Verbesserung des in dieser Arbeit entwickelten eigenen Annotationsprogramms sein.

## 6.4 Übereinstimmungen

Die bislang genutzten Evaluationsmetriken und der in Abschnitt 3.2 vorgestellte  $\kappa$ -Koeffizient können zudem genutzt werden, um die eigenen Annotationsergebnisse mit der Annotation mittels WinMetrix zu vergleichen. Dafür wird der WinMetrix-Output als „Goldstandard“ betrachtet und in Bezug auf diesen wird das eigene Annotationsergebnis evaluiert. Zudem gibt der  $\kappa$ -Koeffizient einen zufallskorrigierten Wert für das Maß der Übereinstimmung zwischen den unterschiedlichen Annotationen an<sup>2</sup>. Die folgenden Werte wurden berechnet:

 **Übereinstimmung zwischen Annotationsergebnissen**  
Untersuchte Verse: 34.599  
Genauigkeit: 0,89  
Abdeckung: 1,00  
F-Maß: 0,94  
 $\kappa$ : 0,88

Es zeigt sich, dass beide Ergebnisse in hohem Maße übereinstimmen. Auch der  $\kappa$ -Koeffizient deutet auf eine hohe Übereinstimmung zwischen den beiden Datensätzen hin. Eine kursorische Analyse der Differenzen zwischen den beiden Algorithmen deutet zudem darauf hin, dass in einigen Fällen fehlender Übereinstimmung der analysierte Vers irregulär, also kein korrekter Hexameter ist: WinMetrix liefert in diesen Fällen vielfach eine verstümmelte Ausgabe, die neue Annotations-Software gibt dagegen – korrekterweise – kein Versmaß aus<sup>3</sup>. Insgesamt deuten die beobachteten Differenzen nicht zwangsläufig auf Annotationsfehler seitens des neuen Annotationsprogramms hin, sondern legen auch Schwächen von WinMetrix (wie etwa die Erzeugung falscher Positive) offen. Korpusverse, denen von den beiden hier betrachteten Algorithmen dasselbe Hexameterschema zugeordnet wurde, können mithin als eher zuverlässig annotiert gelten.

## 6.5 Kennzahlen für die eigene Annotation des Gesamtkorpus

Nicht zuletzt kann im Hinblick auf das neu entwickelte Annotationsprogramm gefragt werden, wieviele potentielle Lösungen dieses für die Korpusverse errechnet.

<sup>2</sup>Als  $\kappa$ -Implementierung wird eine Funktion der Python-Bibliothek *scikit-learn* verwendet: <http://scikit-learn.org/stable/>.

<sup>3</sup>Tatsächlich erhöht sich die gemessene Genauigkeit auf 0,92, wenn diese für die praktische Arbeit mit Hexameterdaten weniger relevanten Fälle nicht berücksichtigt werden, d. h. in 92 % der Fälle, in denen der untersuchte Vers auch tatsächlich mit hoher Wahrscheinlichkeit ein Hexameter ist, liefern beide Algorithmen dasselbe Ergebnis.

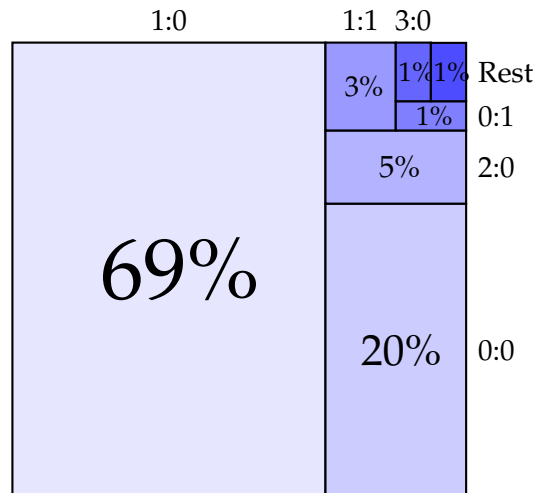


ABBILDUNG 6.2: Anteile der Verse mit einer, mehreren oder keiner Lösung in lokaler+globaler Analyse und Fehlerbehandlung.

Schließlich können auch diese Werte als Indikatoren für die Zuverlässigkeit der Annotation interpretiert werden: Werden beispielsweise für viele Verse mehrere potentielle Lösungen erzeugt, kann dieser Umstand als Indiz für Mängel im linguistischen Regelsatz betrachtet werden, da ein relevanter Anteil der Verse unterbestimmt bleibt, so dass auch der Finite-State Transducer die Verse nicht eindeutig lösen kann.

Der in Abschnitt 4 beschriebene Annotationsalgorithmus setzt sowohl in der globalen Analyse als auch in der Fehlerbehandlung einen FST ein, der jeweils keine oder eine Lösung sowie ggf. mehrere Lösungen erzeugen kann (vgl. Abschnitt 4.4.2.8). Darüber hinaus wird zuvor während der lokalen Analyse eine oder keine (vgl. Abschnitt 4.4.1) Lösung erzeugt. Diese Ergebnisse werden, wie bereits in der Ausgabe des Annotationsprogramms in Abbildung 6.1 ersichtlich (Ausgabe von zwei Integerwerten), in der Folge mit Hilfe der Notation „lokale+globale Analyse : Fehlerbehandlung“ dargestellt. Beispielsweise gibt der Wert 1:0 an, dass während der lokalen und ggf. globalen Analyse eine eindeutige Lösung gefunden wurde. Eine Fehlerbehandlung fand vermutlich nicht statt – falls sie doch stattfand, lieferte sie kein Ergebnis. Abbildung 6.2 stellt das Ergebnis dieser Auswertung grafisch dar. Als Datengrundlage wurde das in Abschnitt 2.4 beschriebene Korpus genutzt.

Erfreulicherweise zeigt die Abbildung, dass das Gros (Werte 1:0 + 1:1 + 0:1 = 73 %) der Verse eindeutig bestimmt wurde. Auch bleibt der Anteil der Verse mit einer Vielzahl möglicher Lösungen gering: Zwar existieren Verse mit drei und mehr Lösungskandidaten, doch machen diese gemeinsam gerade einmal 2 % aller Ergebnisse aus (Werte 3:0 und Rest). Insgesamt erlaubt der genutzte Regelsatz in Zusammenspiel mit dem FST folglich eine eindeutige und – darauf deuten die ebenfalls erfreulichen Evaluationsergebnisse aus Abschnitt 5.4 sowie der Vergleich mit Win-Matrix im vorigen Abschnitt hin – zuverlässige Annotation der meisten Verse. Optimierungen an den Transducer-Gewichten oder die Implementierung einer Auswahlfunktion bei Vorliegen mehrerer Lösungen scheinen derzeit nicht zwingend notwendig zu sein.

Im Vergleich zu diesen Ergebnissen fällt allerdings der Anteil der Verse ohne sicher bestimmte Lösung negativ auf (Wert 0:0, 20 %): Wird durch den FST auch während der Fehlerbehandlung keine Lösung gefunden, wird das Ergebnis der lokalen und ggf. globalen Analyse ausgegeben, sofern dieses ein valides Hexameterschema

ist. Die positiven Evaluationsergebnisse deuten darauf hin, dass durch dieses Vorgehen keine systematischen Annotationsfehler erzeugt werden. Der hohe 0:0-Wert ist aber ein Indiz entweder für Schwächen in den DEAs oder den linguistischen Regeln zur Prüfung der Regelkonformität erzeugter Lösungskandidaten (vgl. z. B. Zeile 22 in Abbildung 4.3): Tatsächlich werden für die Prüfung auf Regelkonformität dieselben – eher konservativen – Regeln genutzt wie für die Identifikation langer Silben (vgl. Abschnitt 4.4.2.6). Eine weitere Analyse muss zeigen, ob durch Relaxierung der Regeln für die Konformitätsprüfung unnötige Korrekturversuche vermieden werden können.

## 6.6 Diskussion

In diesem Abschnitt ging es um Aspekte der praktischen Nutzung der erzeugten Annotationen für althilologische Fragestellungen. Dafür wurde zunächst die Annotationsqualität des Annotationsprogramms WinMetrix von Höflmeier (1982) mit Hilfe des bereits in Abschnitt 5 genutzten Instrumentariums evaluiert, denn unabhängig von möglichen Übereinstimmungen zwischen zwei Algorithmen ist auch die individuelle Güte eines Algorithmus ein wichtiger Indikator für die wissenschaftliche Zuverlässigkeit der genutzten Daten. Es konnte festgestellt werden, dass die durch WinMetrix unter manueller Intervention erzeugten Annotationen eine hohe Zuverlässigkeit aufweisen. Des Weiteren konnte gezeigt werden, dass die neu erzeugte Gesamtannotation des Korpus mit der Ausgabe von WinMetrix in hohem Maß übereinstimmt, so dass durch die doppelte Annotation des Korpus nunmehr ein wissenschaftlich zuverlässig nutzbarer Datensatz entstanden ist. In Bezug auf das im Rahmen dieser Arbeit neu entwickelte Annotationsprogramm hat eine Analyse der Verteilung erzeugter Lösungen über Verse die bisherigen positiven Ergebnisse im Hinblick auf die Qualität der Ausgabe bestätigt, jedoch konnten auch weitere Verbesserungspotentiale aufgezeigt werden.





## Kapitel 7

# Diskussion und Ausblick

Ach! wir kennen uns wenig,  
Denn es waltet ein Gott in uns.

Friedrich Hölderlin, aus dem Gedicht „Der Abschied“

Die vorliegende Arbeit hat sich ausführlich mit dem Problem der automatischen Hexameterannotation beschäftigt. Dafür wurden zunächst in Abschnitt 2 die sprachlichen Besonderheiten der untersuchten Verse diskutiert: Insbesondere wurde aus computerlinguistischer Perspektive auf den sprachlichen Formenreichtum in den Texten der frühgriechischen Epik hingewiesen, der bei der Auswahl eines Annotationsalgorithmus zu berücksichtigen war. Im Anschluss daran wurde in Abschnitt 3 der relevante Forschungsstand analysiert. Auffällig war dabei die Diskrepanz zwischen den üML-Ansätzen aus dem computerlinguistischen Mainstream, die für Anwendungsfälle genutzt werden, die der Hexameterannotation durchaus ähneln, ohne jedoch vollständig deckungsgleich mit ihr zu sein, und der Fokussierung auf regelbasierte Methoden für die Versmaßannotation. Die Gründe hierfür und auch dafür, dass für die eigene Implementierung dennoch ein regelbasierter Ansatz gewählt wurde, sind in diesem Zusammenhang ausführlich diskutiert worden. Zusammenfassend kann an dieser Stelle vermerkt werden, dass die in Abschnitt 3 erwähnten üML-Ansätze unter Nutzung von *n-grams*, Silbeninformationen etc. durchaus auch im Hinblick auf ihre Eignung für die Versmaßannotation erprobt werden sollten – wobei die Verfügbarkeit von Trainingsdaten allerdings eine wesentliche Voraussetzung für entsprechende Untersuchungen darstellt.

Im Anschluss an diese eher theoretischen Ausführungen wurde in Abschnitt 4 das entwickelte Annotationsprogramm dokumentiert und zu zugrundeliegenden abstrakten Konzepten in Beziehung gesetzt, Modellierungsentscheidungen wurden ausführlich begründet. Die gefundene Lösung weist im Vergleich zu bereits vorhandenen Implementierungen einige wesentliche Vorteile auf, gleichwohl wurde aber auch auf Verbesserungspotentiale hingewiesen: Gerade im Hinblick auf komplexe Verse haben sich die (wenigen) linguistischen Regeln, die für die Versmaßannotation genutzt wurden, als unzureichend für eine vollständige Modellierung erwiesen. Der Hypothese, dass datengetriebene Ansätze in diesen komplexen Fällen neue Hinweise zum Verständnis des Problems liefern können – möglicherweise sogar für erfahrene Philologen –, lohnt es sich mithin nachzugehen.

Die Abschnitte 5 und 6 dokumentieren die Evaluation und praktische Anwendung der gefundenen Annotationslösung. Insgesamt wurde eine konkurrenzfähige Annotationsgüte erreicht, die allerdings unter Genauigkeitsgesichtspunkten und im Hinblick auf die Modellierung komplexer Sonderfälle noch weiter verbessert werden kann. Die Nutzung des Finite-State Transducers für die globale Analyse teilannotierter Verse trägt zur erfreulichen Annotationsleistung der Software wesentlich bei, auch wenn keine Möglichkeit bestand, die Transducer-Gewichte aus annotierten

Daten zu erlernen. Im Hinblick auf die praktische Anwendung des Annotationsalgorithmus auf das gesamte Korpus epischer Texte konnte zudem herausgearbeitet werden, dass das neu entwickelte Annotationsprogramm auch den Vergleich mit der bereits langjährig erprobten Implementierung von Höflmeier (1982) nicht scheuen muss. Beide Algorithmen zusammen liefern eine durchaus homogene Gesamtannotation, die neue Perspektiven für das Studium altphilologischer Fragestellungen eröffnet. Allerdings – und dieser Umstand sollte bei der Nutzung der annotierten Daten stets reflektiert werden – hat der Vergleich der Algorithmen auch Hinweise auf Schwachstellen in beiden Programmen geliefert.

Kann das Problem der Hexameterannotation aus computerlinguistischer Sicht als gelöst gelten? Solange noch keine zuverlässige Modellierung für komplexe Phänomene wie Krasis und Synizese vorliegt, ist diese Frage zu verneinen. Beispiele für philologische Fragestellungen, denen auf Grundlage der annotierten Daten nachgegangen werden kann, wurden in Abschnitt 3 gegeben.

# Abbildungsverzeichnis

2.1	Allgemeines Annotationsschema für Hexameter-Verse. . . . .	4
2.2	Systematischer Überblick über Hexameter-Varianten (Höflmeier, 1982, S. 10). . . . .	6
2.3	Schematischer Überblick über die Struktur der von Kruse erzeugten Datenbank (Kruse, 2014, S. 18). . . . .	10
3.1	Beispiele für automatisch mit Finit-State Transducern generierte englische Gedichte aus der Studie von Greene, Bodrumlu und Knight (2010, S. 529.) . . . . .	18
4.1	Zustandsdiagramm eines einfachen DEA (Schulz, 2016, KE 2, S. 5). . .	27
4.2	Finite-State Transducer (Quelle: <a href="https://de.wikipedia.org/wiki/Transduktor_(Informatik)">https://de.wikipedia.org/wiki/Transduktor_(Informatik)</a> ). . . . .	28
4.3	Allgemeiner Annotationsalgorithmus. . . . .	31
4.4	Algorithmus der Prozedur <i>korrigiere</i> für die Fehlerbehandlung. . . .	33
4.5	UML-Paketdiagramm: Komponenten des Annotationsprogramms und Beziehungen zwischen ihnen. . . . .	34
4.6	UML-Klassendiagramm: <i>annotator</i> -Klasse und assoziierte Klassen. . .	35
4.7	<i>preprocessor</i> -Klasse. . . . .	36
4.8	Beispiel für Implementierung linguistischer Regeln: Erkennung von <i>muta cum liquida</i> . . . . .	36
4.9	Programm-Code der Funktion <i>search_long</i> aus der <i>annotator</i> -Klasse. . .	38
4.10	Programm-Code der Prozedur <i>search_whole</i> aus der <i>annotator</i> -Klasse. .	40
4.11	Zustandsdiagramm für den DEA <i>hfsa15</i> . . . . .	41
4.12	Programm-Code der Prozedur <i>search_fourth</i> des DEA <i>hfsa13</i> . . . . .	43
4.13	Gewichteter Finite-State Transducer für die Versmaßvervollständigung (verkürzte Darstellung). . . . .	44
6.1	Ausschnitt aus Gesamtergebnis: erste fünf Verse der Ilias. . . . .	52
6.2	Anteile der Verse mit einer, mehreren oder keiner Lösung in lokaler+globaler Analyse und Fehlerbehandlung. . . . .	54



# Tabellenverzeichnis

2.1	Griechische Diphthonge nach Papakitsos (2011).	7
2.2	Konsonanten und komplexe Konsonantenverbindungen.	8
2.3	Muta cum liquida.	9
2.4	Verteilung der Verse über die einzelnen Subkorpora.	10
3.1	Methodische Einordnung unterschiedlicher Algorithmen.	12
3.2	Zusammenfassende Übersicht über die referierten Ansätze zur automatischen Versmaßannotation.	23
5.1	Überblick über die Evaluationsdatensätze. Für beide Datensätze ist die Verteilung der Verse über die größeren Subkorpora angegeben. Kürzel der Subkorpora entsprechend Tabelle 2.4.	48
5.2	Evaluationsergebnisse Hexameterannotation.	49





# Definitionsverzeichnis

1	Definition (Genauigkeit) . . . . .	12
2	Definition (Abdeckung) . . . . .	12
3	Definition (F-Maß, allgemeine Definition) . . . . .	13
4	Definition (F-Maß, vereinfachte Definition) . . . . .	13
5	Definition (Kappa) . . . . .	13
6	Definition (Deterministischer endlicher Automat) . . . . .	26
7	Definition (Finite-State Transducer) . . . . .	27
8	Definition (Erweiterte Kantenmenge eines FSTs) . . . . .	28
9	Definition (Schnitt eines FSTs und eines endlichen Automaten) . . . . .	29



# Glossar

**agglutinierend** Als agglutinierend werden Sprachen bezeichnet, die „... zum Wortbildungsmittel der Agglutination tendieren ...“. Agglutination ist ein „morphologisches Bildungsprinzip, nach dem ... jedem Morphem ... ein Bedeutungsmerkmal [entspricht], und die Morpheme werden unmittelbar aneinander gereiht, vgl. türk.: *ev* ‚Haus‘, *-im* ‚mein‘, *-ler* ‚Plural‘, *-in* ‚Genitiv‘: *evlerimin* ‚meiner Häuser‘“. 26

**Annotation** „(1) The practice of adding explicit additional information to the machine-readable text; (2) The physical representation of such information.“ (*Corpus Linguistics Glossary*). 1, 7, 9, 11–15, 17, 19–25, 33, 45–48, 51–55

**Daktylus** Der Daktylus ist ein dreisilbiger Versfuß. Die erste Silbe ist im Allgemeinen lang, wohingegen die beiden folgenden Silben in der Regel kurz sind (Bantel und Schäfer, 2000). 5, 17, 43

**Diphthong** „Vokal, bei dem sich während der Artikulation ... auditiv zwei Phasen unterscheiden lassen ...“ (vgl. Tabelle 2.1 für einen Überblick über Diphthonge im Griechischen). 3, 4, 7–9, 22, 61

**Dualis** „Teilkategorie des Numerus zur Bezeichnung von paarweise auftretenden Elementen ....“ 4

**Elision** „Ausfall eines Vokals als Endpunkt eines Vokalabschwächungs-Prozesses ....“ 9, 19, 36

**Hiat** „Auditiv wahrnehmbare Verteilung zweier aufeinander folgender ... Monophthonge auf zwei Silben, z. B. ... in dt. [ˈme:diən] gegenüber umgangssprachlichem [ˈme:ɖɛn] ....“ 7–9, 22, 32, 38, 39

**Hiatkürzung** Eine Hiatkürzung kann bei einem Hiат an der Wortgrenze auftreten: „Ein auslautender Diphthong oder langer Vokal wird [dann] vor [dem anlautenden] Vokal gekürzt.“ (Leggewie u. a., 1981) 9, 50

**Kasus** „Grammatische Kategorie deklinierbarer Wörter, die u. a. zur Kennzeichnung ihrer syntaktischen Funktion im Satz dient ....“ 4

**Kontext** „Als umfassender Begriff der Kommunikationstheorie bezeichnet K. alle Elemente einer Kommunikationssituation, die systematisch das Verständnis einer Äußerung bestimmen: den verbalen und non-verbalen ... K., den aktuellen K. der Sprechsituation und den sozialen K. der Beziehung zwischen Sprecher und Hörer, ihrem Wissen und ihren Einstellungen. Im speziellen Sinn von ‚sprachlicher Umgebung‘ wird neben K. auch der Terminus ‚Kotext‘ ... verwendet.“ 11, 14, 15, 46

**Korpus** „Endliche Menge von konkreten sprachlichen Äußerungen [in maschinenlesbarem Format], die als empirische Grundlage für sprachwiss. Untersuchungen dienen.“ 1–3, 9, 10, 15, 21, 43

**Krasis** Eine Krasis kann bei einem Hiat an der Wortgrenze auftreten: „Auslautender Vokal wird [dann] mit anlautendem Vokal zu einem langen Vokal vereinigt ....“ (Leggewie u. a., 1981) 9, 46, 50, 58

**More** Phonologische Messeinheit für die Länge von Silben. Die Definitionen für ein- und zweimorige Silben im Lateinischen finden sich in Bußmann (1990). 19

**Morphologie** Morphologie „... besteht ... in der Untersuchung von Form, innerer Struktur, Funktion und Vorkommen der Morpheme als kleinsten bedeutungstragenden Einheiten der Sprache.“ 4, 24

**Muta** „Zusammenfassende Bezeichnung der lat.-griech. Grammatik .... Als ‚schweigende‘ Laute unterscheiden sie sich von den Sonanten dadurch, daß sie nicht Silbenträger sein können“ (vgl. Tabelle 2.3 für einen Überblick über muta cum liquida im klassischen Griechisch). 9

**n-gram** „A sequence of  $n$  letters from a given string after removing any spaces ...“ (Baker, Hardie und McEnery, 2006). „Most often,  $n$ -grams in linguistics are sequences of words explored as a unit, where the value of  $n$  denotes how many words are in that unit. If the basic unit of analysis is a word, then we call a word a uni-gram (1-gram). If we have two words to consider as a unit, they are bi-grams (2-grams) ...“ (Crawford und Csomay, 2016).<sup>1</sup> 14, 19, 20, 57

**Numerus** „Grammatische Kategorie des Nomens ... zur Kennzeichnung von Quantitätsverhältnissen.“ 4

**phonetisch** Bezogen auf die Phonetik. „Im Unterschied zur Phonologie untersucht die P. die Gesamtheit der konkreten artikulatorischen, akustischen und auditiven Eigenschaften der möglichen Laute aller Sprachen.“ 8, 15

**phonologisch** Bezogen auf die Phonologie, eine „... Teildisziplin der Sprachwissenschaft, die sich mit den bedeutungsunterscheidenden Sprachlauten ..., ihren ... Eigenschaften, Relationen und Systemen ... beschäftigt.“ 6, 8, 15, 20, 21, 25

**prosodisch** Bezogen auf die Prosodie als „Gesamtheit sprachlicher Eigenschaften wie Akzent, Intonation, Quantität .... Zur P. zählt auch die Untersuchung von Sprechgeschwindigkeit, Rhythmus, Sprechpausen.“ 4, 11, 16, 17, 23

**semantisch** Bezogen auf die Semantik, eine „... Teildisziplin der Sprachwissenschaft, die sich mit der Analyse und Beschreibung der sogen. ‚wörtlichen‘ Bedeutung von sprachlichen Ausdrücken beschäftigt.“ 3, 14, 19, 24, 48

**Spondeus** Metrum der griechischen Lyrik, das aus zwei langen Silben besteht (vgl. Bantel und Schäfer (2000)). 5, 17, 30, 38–40, 42

<sup>1</sup>Bezüglich der Benennungen *uni-gram*, *bi-gram* etc. können sich Manning und Schütze (1999, S. 193, Hervorhebungen im Original) die folgende Bemerkung nicht verkneifen: „[This] will surely be enough to cause any Classicists who are reading this book to stop, and to leave the field to uneducated engineering sorts: *gram* is a Greek root and so should be put together with Greek number prefixes. Shannon actually *did* use the term *digram*, but with the declining level of education in recent decades, this usage has not survived.“

**Subkorpus** A sub corpus is „a component of a corpus, usually defined using certain criteria such as text types and domains“ (*Corpus Linguistics Glossary*). 10, 49, 51

**Synizese** Eine Synizese kann bei Hiat im Wortinneren auftreten: „Zwei zusammenstoßende Vokale, die an und für sich den Wert zweier Silben haben sollten, rücken [dann] zu einer Silbe zusammen, ohne daß dies durch die Schrift zum Ausdruck gebracht wird ....“ (Leggewie u. a., 1981) 9, 32, 46, 51, 58

**syntaktisch** Bezogen auf die Syntax, einen „Teilbereich der Grammatik natürlicher Sprachen ...: System von Regeln, die beschreiben, wie aus einem Inventar von Grundelementen ... durch spezifische syntaktische Mittel ... alle wohlgeformten Sätze einer Sprache abgeleitet werden können.“ 6, 14, 15, 24, 26

**Tagging** „A more informal term for the act of applying additional levels of annotation to corpus data. A tag usually consists of a code, which can be attached to a phoneme, morpheme, word, phrase or longer stretch of text in a number of ways. ... Tagging is often carried out automatically using software ....“ (Baker, Hardie und McEnery, 2006). 14

**Varietät** „Neutraler Terminus für eine bestimmte kohärente Sprachform, wobei spezifische außersprachliche Kriterien varietätendefinierend eingesetzt werden können ....“ 16, 22, 24

Die Definitionen des Glossars entstammen, soweit nicht anders angegeben, der Arbeit von Bußmann (1990).



# Literatur

- Agirrezabal, Manex u. a. (2016). „ZeuScansion: A tool for scansion of English poetry“. In: *Journal of Language Modeling* 4.1, S. 3–28.
- Artstein, Ron und Massimo Poesio (2008). „Inter-Coder Agreement for Computational Linguistics“. In: *Computational Linguistics* 34.4, S. 555–596.
- Baker, Paul, Andrew Hardie und Tony McEnery (2006). *A Glossary of Corpus Linguistics*. Edinburgh University Press.
- Bantel, Otto und Dieter Schäfer (2000). *Grundbegriffe der Literatur*. 16. Auflage. Cornelsen.
- Baumann, Timo und Burkhard Meyer-Sickendiek (2016). „Large-scale Analysis of Spoken Free-verse Poetry“. In: *Proceedings of the Workshop on Language Technology Resources and Tools for Digital Humanities (LT4DH), COLING 2016*. The COLING 2016 Organizing Committee, S. 125–130.
- Beesley, Kenneth R. und Lauri Karttunen (2003). *Finite State Morphology*. CSLI Studies in Computational Linguistics. CSLI Publications.
- Beierle, Christoph, Gabriele Kern-Isberner und Manfred Widera (2009). *Wissensbasierte Systeme*. Kurstext. Fernuniversität Hagen.
- Bußmann, Hadumod (1990). *Lexikon der Sprachwissenschaft*. 2. Auflage. Kröner.
- Ciobanu, Alina Maria, Anca Dinu und Liviu P. Dinu (2014). „Predicting Romanian Stress Assignment“. In: *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2014)*. Association for Computational Linguistics, S. 64–68.
- Cohen, Jacob (1960). „A coefficient of agreement for nominal scales“. In: *Educational and Psychological Measurement* 20.1, S. 37–46.
- Corpus Linguistics Glossary*. Sammlung von Begriffsdefinitionen zur Korpuslinguistik auf der Internet-Seite des Instituts für Angewandte Linguistik der Kent State University in Kent, Ohio, USA. URL: <https://www.kent.edu/appling/corpus-linguistics-glossary>.
- Crawford, William J. und Eniko Csomay (2016). *Doing Corpus Linguistics*. Routledge.
- Estes, Alex und Christopher Hench (2016). „Supervised Machine Learning for Hybrid Meter“. In: *Proceedings of the Fifth Workshop on Computational Linguistics for Literature, NAACL-HLT 2016*. Association for Computational Linguistics, S. 1–8.
- Greene, Erica, Tugba Bodrumlu und Kevin Knight (2010). „Automatic Analysis of Rhythmic Poetry with Applications to Generation and Translation“. In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP 2010)*. Association for Computational Linguistics, S. 524–533.
- Hackstein, Olav (2011a). „Der sprachhistorische Hintergrund“. In: *Homer Handbuch: Leben – Werk – Wirkung*. J. B. Metzler, S. 32–45.
- (2011b). „Homerische Metrik“. In: *Homer Handbuch: Leben – Werk – Wirkung*. J. B. Metzler, S. 26–32.
- Hench, Christopher (2017a). „Phonological Soundscapes in Medieval Poetry“. In: *Proceedings of the Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Science, Humanities and Literature, ACL 2017*. Association for Computational Linguistics, S. 46–56.



- Hench, Christopher (2017b). „Prosodic Clustering via Cosine Similarity of Sound Sequence Inventories“. In: *Proceedings of the Digital Humanities Conference 2017*.
- Höflmeier, Johann (1982). „Metrisches zum frühgriechischen Epos“. Unveröffentlichter Forschungsbericht.
- Hölderlin, Friedrich (1977). *Gedichte*. 4. Auflage (Entstehung der zitierten Gedichte zwischen 1799–1806). Reclam.
- Karttunen, Lauri und Kenneth R. Beesley. *A Short History of Two-Level Morphology*. URL: <http://www.ling.helsinki.fi/~koskenni/esslli-2001-karttunen/>.
- Koskeniemi, Kimmo (1983). „Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production“. Diss. Universität Helsinki. URL: <http://www.ling.helsinki.fi/~koskenni/doc/Two-LevelMorphology.pdf>.
- Kruse, Sebastian (2014). „Entwicklung eines web-basierten Systems zur Unterstützung der Suche nach Parallelstellen in Homers Epen mit Hilfe sprachstatistischer Methoden“. Magisterarb. Fernuniversität Hagen.
- Larsen-Freeman, Diane und Lynne Cameron (2008). *Complex Systems and Applied Linguistics*. Oxford University Press.
- Leggewie, Otto u. a. (1981). *Ars Graeca: Griechische Sprachlehre*. Ferdinand Schöningh.
- Manning, Christopher D. und Hinrich Schütze (1999). *Foundations of Statistical Natural Language Processing*. The MIT Press.
- Mannsperger, Brigitte und Dietrich Mannsperger (2017). „Versmaß, Sprachform und Wortwahl“. In: *Homer verstehen*. Aktualisierte Sonderausgabe. Wissenschaftliche Buchgesellschaft, S. 27–34.
- McCurdy, Nina, Vivek Srikumar und Miriah Meyer (2015). „RhymeDesign: A Tool for Analyzing Sonic Devices in Poetry“. In: *Proceedings of the Fourth Workshop on Computational Linguistics for Literature, NAACL 2015*. Association for Computational Linguistics, S. 12–22.
- Meyer-Sickendiek, Burkhard, Hussein Hussein und Timo Baumann (2017). „Rhythmicalizer: Data Analysis for the Identification of Rhythmic Patterns in Readout Poetry (Work-in-Progress)“. In: *INFORMATIK 2017. Lecture Notes in Informatics (LNI)*. Gesellschaft für Informatik, S. 2189–2200.
- Navarro-Colorado, Borja (2015). „A computational linguistic approach to Spanish Golden Age Sonnets: metrical and semantic aspects“. In: *Proceedings of the Fourth Workshop on Computational Linguistics for Literature, NAACL 2015*. Association for Computational Linguistics, S. 105–113.
- Oflazer, Kemal (1994). „Two-level Description of Turkish Morphology“. In: *Literary and Linguistic Computing* 9.2, S. 137–148.
- Papakitsos, Evangelos C. (2011). „Computerized Scansion of Ancient Greek Hexameter“. In: *Literary and Linguistic Computing* 26.1, S. 57–69.
- Pavese, Carlo Odo und Federico Boschetti (2003). „Introduction: Description of the Programme. Directions for the Formular Edition“. In: *A Complete Formular Analysis of the Homeric Poems*. Adolf M. Hakkert, Band 1.
- Rengakos, Antonios und Bernhard Zimmermann (2011). *Homer Handbuch: Leben – Werk – Wirkung*. J. B. Metzler.
- Reynolds, Robert und Francis Tyers (2015). „Automatic word stress annotation of Russian unrestricted text“. In: *Proceedings of the 20th Nordic Conference of Computational Linguistics (NODALIDA 2015)*. Linköping University Electronic Press, S. 173–180.
- Roche, Emmanuel und Yves Schabes (1997a). *Finite-State Language Processing*. Language, Speech, and Communication. The MIT Press.

- (1997b). „Introduction“. In: *Finite-State Language Processing*. The MIT Press, S. 1–66.
- Rösler, Wolfgang (2011). „Mündlichkeit und Schriftlichkeit“. In: *Homer Handbuch: Leben – Werk – Wirkung*. J. B. Metzler, S. 201–213.
- Schulz, André (2016). *Grundlagen der Theoretischen Informatik A*. Kurstext. Fernuniversität Hagen.
- Strasser, Franz Xaver (1984). *Zu den Iterata der frühgriechischen Epik*. Beiträge zur Klassischen Philologie 156. Verlag Anton Hain.
- Sültmann, Jonas (2018). „Eine metrische Analyse des Corpus der frühgriechischen Epik“. Freie Universität Berlin. Unveröffentlichte Hausarbeit am Institut für Griechische und Lateinische Philologie.
- Tyers, Francis M. u. a. (2017). *Third International Workshop on Computational Linguistics for Uralic Languages: Proceedings of the Workshop*. Association for Computational Linguistics.
- Yli-Jyrä, Anssi, Lauri Karttunen und Juhani Karhumäki (2006). *Finite-State Methods and Natural Language Processing: 5th International Workshop, FSMNLP 2005*. Lecture Notes in Artificial Intelligence 4002. Springer.
- Zabaletak, Manex Aguirrezabal (2017). „Automatic Scansion of Poetry“. Diss. Universität des Baskenlandes (Euskal herriko unibertsitatea).