Korpus im Text

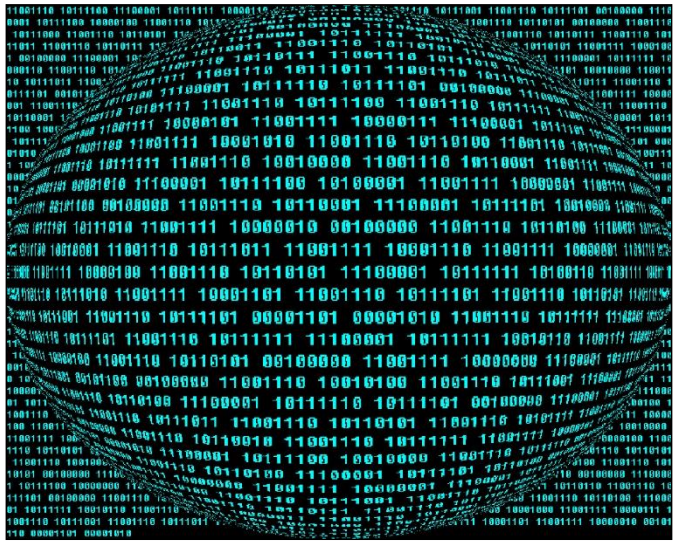
Stephan Lücke
Christian Riepl
Caroline Trautmann

Softwaretools und Methoden
für die korpuslinguistische Praxis

1

Korpus im Text

Band 1



Korpus im Text

Herausgegeben

von

Thomas Krefeld

Stephan Lücke

Christian Riepl

2017

Abbildung auf der Titelseite: Der Buchstabe M als Initiale des Wortes „mortuus“ (Lukasevangelium, 16, 22) im Book of Kells (entstanden um 800; folio 254v; Trinity College Dublin; Aufnahme aus Schautafel: Stephan Lücke 2014)

Frontispiz: Die Verse 1-7 der Ilias, kodiert nach Unicode und UTF-8 in binärer Gestalt, erzeugt mit dem Unix-Kommando xxd und verfremdet mit dem Programm Gimp (Quelle des Basistextes: http://www.perseus.tufts.edu/hopper/text?doc=Perseus:text:1999.01.0133)

Softwaretools und Methoden für die korpuslinguistische Praxis

von

Stephan Lücke
Christian Riepl
Caroline Trautmann

2017

Herausgegeben von der
Universitätsbibliothek der Ludwig-Maximilians-Universität
Geschwister-Scholl-Platz 1
80539 München

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie. Detaillierte bibliografische Daten sind im Internet unter http://dnb.dnb.de abrufbar.

Open-Access-Version dieser Publikation verfügbar unter:
https://doi.org/10.5282/kit01
http://nbn-resolving.de/urn:nbn:de:bvb:19-epub-36308-6
ISBN: 978-3-95896-016-9 (elektronische Version)

Editorial

Der Titel dieser Publikationsplattform ist Programm, denn sie wendet sich an die Wissenschaftler, die eine direkte Verschränkung von sprachlicher Darstellung und Datenbasis suchen. In Printpublikationen ist ein solches Umschalten zwischen den beiden Dimensionen wissenschaftlicher Arbeit, wenn überhaupt nur sehr bedingt möglich (etwa in Form eines Begleitbandes). Die hypertextuelle Verlinkung befreit dagegen das Korpus aus seiner Isolierung und macht es zum gleichberechtigten Gegenstand der Aufmerksamkeit. Besonders sinnvoll ist diese Emanzipation, wenn die genuinen Daten nicht nur sprachlicher bzw. im Sprachlichen nicht schriftlicher Natur sind. Weiterhin ist es möglich, dynamische Korpora mit wachsenden Datenbeständen einzubetten. Erst in diesem medialen Format, das über die Präsentation einer beschriebenen Fläche – sei es auf einem Blatt Papier oder auf einem Bildschirm – weit hinausgeht, kann das e-Book sein eigentliches Potential entfalten.

Thomas Krefeld | Stephan Lücke | Christian Riepl

Inhaltsverzeichnis

Vorwort

Der vorliegende erste Band der KiT-Reihe bedarf in doppelter Hinsicht zumindest einer Erläuterung, eher noch einer Rechtfertigung. Denn obwohl eigentlich der erste Band der Reihe, erscheint er doch zeitlich nach deren Band 2[1], und darüberhinaus entspricht er im Hinblick auf die gewählte Publikationsform ganz und gar nicht der programmatischen Absicht der KiT-Herausgeber, die mit der Reihe im Umfeld des Wissenschaftsbetriebs ganz neue und zeitgemäße Formen der Publikationskultur etablieren möchten. Der eine Mangel bedingt den anderen: Mit der Abfassung des Textes wurde bereits im Jahr 2011 begonnen, zu einer Zeit als der zentrale Gedanke der KiT-Reihe noch nicht geboren war. Die vielfältigen Verpflichtungen der Autoren verhinderten eine zügige Fertigstellung, so dass sich gerade angesichts der Veränderung des Publikationsbegriffs schließlich die Frage stellte, ob eine Publikation in der ursprünglich angelegten Gestalt überhaupt noch vertretbar ist. Wir haben uns letztlich dazu entschieden, den Band dennoch in der hier vorliegenden konventionellen, auf dem Konzept der Buchseite beruhenden Gestalt zu veröffentlichen, und planen, in kurzer Zeit eine aktualisierte Version vorzulegen, die dann konsequent den editorischen Idealen der KiT-Reihe entsprechen wird. Auf diese Weise erlangt der erste Band der Reihe programmatischen Charakter und steht symbolisch für den Übergang von der alten hin zu einer neuen Publikationskultur im Bereich der Wissenschaft.

Seit dem Sommersemester 2009 betreuen die Autoren im Rahmen ihrer Lehr- und Mentorentätigkeit an der LMU München Doktoranden und Studenten der Sprachwissenschaften, die sich auf das Gebiet der Korpuslinguistik spezialisieren und die zur Erstellung von digitalen Korpora, deren Bearbeitung und Auswertung Methoden der praktischen Informatik einsetzen möchten. Obwohl das Internet für die Arbeit mit Korpora Quellen, Programme und ↗Datenbanken sowie Handbücher und Ratgeber relativ zahlreich anbietet, sich ständig ergänzt und weiter vernetzt, fehlt es doch an einer Hilfe, die in einem konkreten Forschungsvorhaben informatische Verfahrensweisen zur Behandlung eines Gegenstandes systematisch darlegt und methodisch begründet. Es sei an dieser Stelle darauf hingewiesen, dass sich das vorliegende Werk ausschließlich als Handreichung für die praktische Arbeit des Korpuslinguisten versteht. So ist zu erklären, warum der Band keine Bibliogra-

[1] Thomas Krefeld, Stephan Lücke, Emma Mages (Hrsg.) (2015): Zwischen traditioneller Dialektologie und digitaler Geolinguistik: Der Audioatlas siebenbürgisch-sächsischer Dialekte (ASD). Korpus im Text; Bd. 2. Münster: Monsenstein & Vannerdat

phie aufweist und entsprechend auch keine Auseinandersetzung insbesondere mit dem seit einigen, wenigen Jahren sich entwickelnden Bereich der Sekundärliteratur zum Thema Digital Humanities (DH) erfolgt.[2]

Ausgangspunkt sind Problemstellungen einzelner Dissertations- und in Zukunft auch Bachelor- und Masterprojekte, die im Rahmen von Lehrveranstaltungen im Themenbereich Beschreibung, Abbildung und Visualisierung von Sprache und Text behandelt wurden. Die daran erarbeiteten Problemlösungen, die Wahl bestimmter Softwarewerkzeuge und deren Funktionsweise sowie der Einsatz informatischer Methoden im Bereich der Korpuslinguistik werden systematisch zusammengestellt und erklärt. Das Handbuch ist veranstaltungsbegleitend zum Nachschlagen und selbstständigen Nacharbeiten gedacht. Es soll in die Lage versetzen, die erhobenen Daten so zu organisieren und zu verwalten, dass sie für die wissenschaftliche Fragestellung und Auswertung optimal aufbereitet sind und im Internet veröffentlicht bzw. zur Verfügung gestellt werden können.

Die im Handbuch vermittelten IT-Kenntnisse beruhen auf langjährigen Erfahrungen im Umgang mit digitalen Korpora, deren rechnergestützter Analyse und Auswertung unter fachspezifischen Aspekten. Die Auswahl der hier vorgeführten Software orientierte sich zunächst an der Verbreitung der verwendeten ↗Betriebssysteme – meist Microsoft Windows, seltener Apple MacOS oder Unix/Linux – maßgeblich waren sodann die problemlose Installation der Softwarekomponenten sowie deren einfache und praktische Bedienbarkeit, ferner ihre modulare und flexible Anwendbarkeit und schließlich – idealerweise – ihre freie Verfügbarkeit. Bei der Zeichenkodierung, der Mustererkennung oder dem Datenbankentwurf werden bewährte Konzepte übernommen. Auf den Gebieten der Datenstrukturierung und Programmierung sind die Kriterien Plattformunabhängigkeit, Kompatibilität, Konvertierbarkeit und Anpassungsfähigkeit auch im Hinblick auf die langfristige Speicherung

[2] Stellvertretend für viele andere Publikationen aus dem Bereich der Digital Humanities seien hier nur die folgenden genannt: Gardiner, Eileen / Musto, Ronald G., The digital humanities: a primer for students and scholars. Cambridge University Press, 2015; Kurz, Susanne. Digital Humanities: Grundlagen und Technologien für die Praxis. Springer-Verlag, 2016; Jannidis, Fotis / Kohle, Hubertus / Malte Rehbein (Hrsgg.), Digital Humanities: Eine Einführung. Springer-Verlag, 2017. Erwähnt sei schließlich auch noch Franco Moretti, dessen Buch "Distant reading" (Verso Books, 2013) in den letzten Jahren einige Aufmerksamkeit erregt hat, jedoch mit seiner eher literaturwissenschaftlichen Perspektive in eine deutlich andere Richtung geht als der hier vorgelegte Text oder auch die anderen oben zitierten Titel.

der Daten und ihrer nachhaltigen Verwendung maßgebend. In der Summe geht es nicht um die Präsentation eines fertigen, kompakten und universal einsetzbaren Softwareprodukts. Vielmehr werden den mit digitalen Materialien arbeitenden Korpuslinguisten effizient zu handhabende Softwarewerkzeuge zugänglich gemacht, die je nach Bedarf auf den einzelnen Gegenstand und das Forschungsziel ausgerichtet sind und eine wohl strukturierte Datenbasis für die linguistische Auswertung sowohl in der konkreten Forschungsthematik als auch unter zukünftigen Fragestellungen erschließen.

Wir danken allen Absolventen des Linguistischen Internationalen Promotionsprogrammes (LIPP; nunmehr „Graduate School Language & Literature Munich — Class of Language“) an der LMU, die unsere Lehrveranstaltungen während der letzten Jahre besucht und durch viele interessante Fragen und Probleme zum Entstehen dieses Handbuchs beigetragen haben. Unser Dank gilt auch Daniel Wasöhrl und Iryna Taranenko, die den Text kritisch durchgesehen haben, sowie Frau Yan Peng für ihre Mitarbeit bei der Vorbereitung des Textes zur Publikation.

München, im Februar 2017

Stephan Lücke | Christian Riepl | Caroline Trautmann

1 Einleitung

In diesem Handbuch wird vermittelt, wie Sprachdaten in gesprochener oder geschriebener Form für korpusbasierte linguistische Forschungsarbeiten (besonders Dissertationen) sinnvoll und systematisch erhoben und so weiterverarbeitet werden können, dass sie schließlich in eine relationale (z.B. MySQL-) ↗Datenbank übertragen werden können. Die ↗Datenbank ermöglicht komplexe Abfragen, welche die für die Beantwortung der Forschungsfragen erforderlichen Angaben liefern. Eine ↗Datenbank erlaubt auch stets die spätere Anlagerung von weiteren ↗Metadaten bzw. die freie Kombination mit nahezu beliebigen anderen Daten zu Zwecken, die über das ursprüngliche Forschungsinteresse hinausgehen und für andere Forschungen genutzt werden können.

Eine sorgfältige, reflektierte Vorbereitung der Datenerhebung und -strukturierung beschränkt Fehler und Kompromisse, die sich aus der konkreten Situation im Feld oder auf Grund einer bestimmten Quellenlage zwangsläufig ergeben, auf das unvermeidbare Minimum. Insbesondere kann eine durchdachte Vorbereitung vermeiden helfen, dass die bearbeitbaren Forschungsfragen wegen der Eigenschaften des Materials eingeschränkt werden. Es soll der Weg beschritten werden, die Datenerhebung von vorne herein so reflektiert zu gestalten, dass möglichst alle sinnvollen Forschungsfragen und Hypothesen bearbeitbar sind.

Sprachliche Daten können Ton- bzw. Video-Daten oder Textdaten sein. Sie können – und das ist die hier primär ins Auge gefasste Vorgehensweise – vom Forscher selbst für die Zwecke des jeweiligen Forschungsprojektes erhoben werden, oder es handelt sich um Daten, die ihrerseits schon aus einem Korpus wie z.B. dem British National Corpus[3] (BNC) extrahiert sind, also um Auszüge von bereits bestehenden, umfangreicheren Korpora.

Um die Erhebung der Daten und ihre anschließende Strukturierung sinnvoll und möglichst wenig fehlerträchtig zu organisieren, ist es erforderlich, zum Zeitpunkt der Datenerhebung, spätestens aber zum Zeitpunkt der Datenstrukturierung Entscheidungen über die Forschungsfragen zu treffen, die mit Hilfe der Daten beantwortet werden

[3] Sämtliche im Text zitierten Internet-Links wurden im März 2017 noch einmal überprüft. Es liegt in der Natur der Sache, dass die Fortexistenz der entsprechenden Webseiten und die Persistenz der Links nicht garantiert werden können.

sollen; ebenso müssen die dafür notwendigen analytischen Kategorien festgelegt sein. Die zielgenaue Erhebung der Daten und ihre Aufbereitung ist also ein Teil der analytischen Arbeit. Dies beginnt schon mit sehr kleinen Details, wie in den beiden folgenden Beispielen beschrieben.

Beispiel 1: Werden sprachliche Daten in der Feldforschung erhoben, muss vor dem Beginn der Erhebung geklärt sein, welche sozialen und biographischen Merkmale der Informanten für die Analyse relevant sind. Bei der Datenerhebung selbst werden diese Merkmale mit erhoben. Bei der Datenstrukturierung werden sie in die Dateinamen mit eingearbeitet, so dass später ganz gezielt Daten von Teilgruppen der Informanten abgefragt werden können.

Wird mit bereits bestehenden Daten gearbeitet, muss von vorne herein darauf geachtet werden, ob alle für das Forschungsprojekt notwendigen Angaben (↗Metadaten, Annotationen) verfügbar sind. Neben inhaltlichen Fragen, sind schon in der frühen Phase der Datengewinnung und -erhebung auch scheinbar periphere technische Aspekte zu berücksichtigen. Wird dies unterlassen, entsteht möglicherweise ein ganz erheblicher Nachbearbeitungsbedarf oder gar „Datenmüll", der für eine systematische Bearbeitung nicht mehr brauchbar ist.

Beispiel 2: Werden Daten aus dem Internet gewonnen und in Text-Dateien gespeichert bzw. per copy-paste-Verfahren in Textdateien eingegeben, so sollten alle Daten einheitlich kodiert sein oder müssen in eine einheitliche Kodierung gebracht werden, bevor sie zu einem Korpus zusammengefügt werden und mit weiteren Tools bearbeitet werden. Nur im (unwahrscheinlichen) Fall, dass alle Texte ausschließlich ↗ASCII-kodiert sind, können sie bedenkenlos zusammengefügt werden.

Folgende Schritte der Datenerhebung und –strukturierung sind in der Regel nacheinander abzuarbeiten:

1. Planung des Forschungsvorhabens (z. B. in Form eines elaborierten Exposés)
2. Konzeption der Datenerhebung bzw. Auswahl der Daten, Recherche zur Ausgangsbeschaffenheit der Rohdaten
   a. Audio-Daten
   b. Video-Daten
   c. Schriftliche Daten (digital oder papieren, gedruckt oder Manuskripte)

   d. ↗Metadaten wie soziale und biographische Daten oder Notizen aus Beobachtungen, Fragebögen o.ä.

3. Auswahl verlässlicher und zeitökonomischer manueller und technischer Bearbeitungsverfahren
   a. Transkriptionen von Audio-/Video-Daten (orthographisch/literarisch/phonetisch) mit einem Editor oder mit dem Programm Praat
   b. Transkription von Manuskripten
   c. Verschriftlichung von Notizen etc.
   d. Digitales Erfassen und Ablegen von digitalen Texten unterschiedlicher Herkunft (Web, digitale Textsammlungen, Korpora, etc.)
   e. Sonderfall: Digitalisierung von Texten mit Hilfe von OCR (Optical Character Recognition, wie z.B. ABBYY Finereader) und gegebenenfalls Nachbearbeitung
4. Erstellung einer Fassung von „gesäuberten" Rohdaten, gegebenenfalls annotiert (z.B. in Praat)
   a. Überführung der Rohdaten in ein einheitlich kodiertes (minimalistisches) Textformat und Entfernung bzw. Umkodierung aller Zeichen aus den Dateien, die nicht zu den Sprachdaten gehören (graphische Merkmale wie Fettdruck, verschiedene Schriftarten, überschüssige Zeilenumbrüche etc.)
   b. Bei aus Texten gewonnenen Daten (digital und analog): Aufsplittung von Daten und ↗Metadaten/Annoationen nach einem einheitlichen und transparenten Verfahren zur getrennten Weiterverarbeitung
5. Automatisierte Zerlegung der Daten in datenbankfähige Einheiten mit Hilfe von geeigneten Tools
   a. Zerlegung in Sätze / Äußerungen
   b. Tokenisierung
   c. Zerlegung in andere, der Fragestellung angemessene Einheiten.
      WICHTIG: jeweils mit eindeutiger Referenzierung auf das Ausgangsmaterial über IDs, Satznummern, Tokennummern etc.
6. Annotation der so gewonnenen Elemente, z.B.

   a. morphologische Kategorien (↗Lemma, Wortart/Part of Speech, Numerus, Genus, etc.)
   b. syntaktische Kategorien (Syntagmen, Circumstantialien, Satzbaupläne, etc.)
7. Import der so vorbereiteten Daten in eine relationale (z.B. MySQL-) ↗Datenbank

Die Arbeitsschritte 1. und 2. werden im Handbuch nicht weiter behandelt. Der Einsatz informatischer Methoden zur Bearbeitung der Daten beginnt mit dem Arbeitsschritt 3. Die technische Bearbeitung der Daten soll immer so erfolgen, dass eine eindeutige Zuordnung der weiterverarbeiteten Daten zu den Rohdaten möglich ist. Alle Informationen, die von den eigentlichen Daten abgelöst werden, die für die Analyse jedoch wichtig sind oder werden können, müssen separat vorgehalten werden. Hierzu können beispielsweise auch graphische Informationen gehören, falls gedruckte Texte bearbeitet werden, die Informationen mit graphischen Mitteln bereitstellen (Überschriften, Fußnoten etc.).

Die Aufarbeitung der Rohdaten hin zu einer solchen ↗Datenbank ermöglicht es den Forschern auch, ihr Material im Detail kennenzulernen, um so nach der erfolgten Datenstrukturierung unmittelbar in die Analyse einsteigen zu können.

Im Sinne der Operationalisierung und zur transparenten Planung von Workflows zur Digitalisierung von Textquellen haben wir an der IT-Gruppe Geisteswissenschaften (ITG) der Ludwig-Maximilians-Universität München (LMU) ein Schema entworfen, das verschiedene Digitalisierungsgrade unterscheidet sowie Phasenübergänge, Annotierungs- und Analyseprozeduren sowie Exportformate definiert:

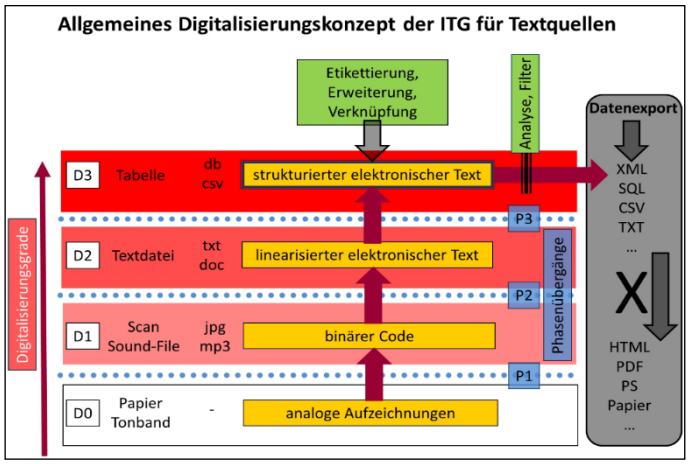


Abbildung 1: Allgemeines Digitalisierungskonzept für Textquellen

Im Handbuch werden zunächst die informatisch grundlegenden Werkzeuge und Konzepte vorgestellt (Kap. 3). Diese können in allen o.g. Bearbeitungsschritten zum Einsatz kommen. Kap. 4 befasst sich mit Verfahren zur Digitalisierung von gesprochener Sprache und gedrucktem Text, Kapitel 5 schließlich beschreibt das relationale Datenmodell und die darauf aufbauenden Analyseverfahren.

Im Anhang (S. 187ff.) findet man ein Glossar[4] (S. 188), eine Checkliste zur Vermeidung unnötiger Probleme (S. 187) sowie diverse Dokumentationen (z.B. eine ↗ASCII-Tabelle [S. 199], ein Schaubild zur ↗UTF-8-Kodierung [S. 202] etc.), Codebeispiele und Konfigurationsdateien für die vorgestellten Softwaretools.

Mit dem AsiCa-Projekt soll im Folgenden zunächst ein Beispiel aus der Forschungspraxis zur Einführung in die Thematik gegeben werden.

[4] Verweise auf Einträge im Glossar erfolgen durch den „NORTH EAST ARROW (U+2197)“ (↗).

2 Ein Beispiel aus der Praxis: Das AsiCa-Projekt

An der Ludwig-Maximilians-Universität München wurde seit 2004 unter der Leitung von Prof. Thomas Krefeld der „Atlante Sintattico della Calabria“, kurz „AsiCa“, erarbeitet. Es handelt sich um ein umfangreiches Korpus gesprochener Sprache, dass das Sprachverhalten von Sprechern des süditalienischen Dialekts „Calabrese“ vor dem Hintergrund von Migrationsbewegungen dokumentiert und analysiert.

Die Präsentation dieses Korpus ist der Beschreibung der für Korpuslinguisten relevanten Computer-Technologien vorangestellt, um gleichsam an einem „fertigen Produkt“ exemplarisch darzulegen, was bei geschickter Konzeption und kompetenter Beherrschung der nötigen Werkzeuge entstehen bzw. erreicht werden kann. Jedem Einzelnen ist es überlassen, die hier vorgetragenen Prinzipien an die eigenen Erfordernisse anzupassen und entsprechend umzusetzen.

Das AsiCa-Korpus ist im Internet konsultierbar: http://asica.gwi.uni-muenchen.de/.[5]

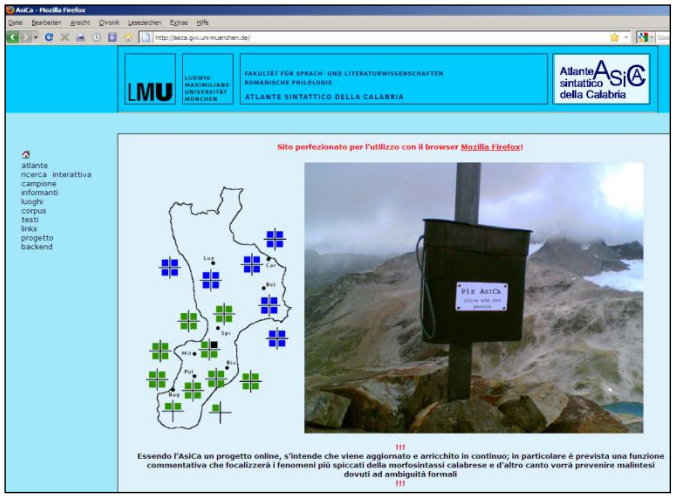


Abbildung 2: Homepage des Atlante sintattico della Calabria (AsiCa)

[5] Seit Februar 2017 sind AsiCa je ein persistenter „Digital Object Identifier“ (DOI: http://doi.org/10.5282/asica) und ein „Uniform Resource Name“ (URN: http://nbn-resolving.de/urn:nbn:de:bvb:19-asica-5) zugeordnet. Zu Konzept und Funktionsweise von DOI bzw. URN s. Lücke, S.: s.v. “Digital Object Identifier (DOI)”, in: VA-de 16/2, Methodologie, https://www.verba-alpina.gwi.uni-muenchen.de/?page_id=493&db=162&letter=D#73.

Bestimmte Bereiche dieser Webseite sind aus Gründen des Schutzes von Persönlichkeitsrechten paßwortgeschützt. Das Paßwort wird auf Anfrage und ausschließlich zu wissenschaftlichen Zwecken an Interessierte herausgegeben.

Im Wortsinne grundlegend für die Zuverlässigkeit der Ergebnisse der Datenanalyse ist das Vorhandensein einer strukturierenden Systematik von Anbeginn, d.h. noch vor der Datenerhebung. Im Fall des AsiCa-Korpus ergab sich die Systematik aus dem methodischen Ansatz, das Sprechverhalten von aus Kalabrien stammenden und in Deutschland lebenden Migranten mit dem Sprechverhalten von ortsfesten Nicht-Migranten in Kalabrien zu vergleichen. Damit war bereits eine erste grobe Gliederung des noch zu erhebenden Sprachmaterials angelegt und entsprechend die Auswahl der Informanten prädeterminiert. Zusätzlich zu dieser ersten Grobgliederung wurden – basierend auf der Hypothese, dass sich diese Merkmale im jeweiligen Sprachgebrauch widerspiegeln würden – weitere Merkmale definiert, an denen sich die Auswahl der Informanten zu orientieren hatte: so wurde unterschieden zwischen Männern und Frauen, Eltern und Kindern sowie den kalabresischen Heimatorten der Informanten. Bei der Auswahl der Heimatorte wurde auf eine gleichmäßige Verteilung in der Fläche sowie auf das zumindest postulierbare Vorhandensein eines möglichst ungestörten kalabresischen Dialekts geachtet. Noch vor Beginn der Materialerhebung wurde weiterhin eine Unterscheidung zwischen der gezielten Evozierung von Sprache auf Basis eines Fragebogens (ähnlich dem von Georg Wenker) und der spontanen Sprachproduktion getroffen. Damit waren Umfang und Inhalt des Korpus bereits vor seiner Entstehung festgelegt: Es sollte sowohl gelenkt evoziertes wie auch spontansprachliches Material von Informanten aus insgesamt acht Ortschaften in Kalabrien enthalten, von denen, bezogen auf jeden einzelnen Ort, jeweils genau gleich viele den folgenden Gruppen zuzuordnen wären: Männer/Frauen, Migranten/Ortsfeste, Eltern/Kinder. Ein weiteres Postulat war gewesen, dass sämtliche jeweils aus einem Ort stammende Informanten, zu jeweils ein und derselben Familie gehören mussten.

Abstrakt gesprochen, läßt sich das beschriebene System als eine Reihe von Merkmalen begreifen, von denen jedes jeweils unterschiedliche Merkmalsausprägungen annehmen kann:

Merkmal	Ausprägungen	Anzahl der Ausprägungen
Herkunftsort	ort 1, ort 2, ort 3 ...	8
Geschlecht	weiblich, männlich	2
Generation	Eltern, Kinder	2
Migrant	ja, nein	2
Produktion	gelenkt, spontan	2

Bei genauer Betrachtung beziehen sich die Merkmale auf unterschiedliche „Gegenstände" (Entitäten): Während „Herkunftsort", „Geschlecht", „Generation" und „Migrant" Merkmale der Entität „Informant" sind, bezieht sich „Produktion" auf die Entität „Sprachäußerung".

Die Anzahl der Informanten, die zu interviewen waren, ergibt sich aus der Multiplikation der jeweiligen Anzahl der Merkmalsausprägungen, also: 8*2*2*2=64.

Da jeder der Informanten sowohl ein spontansprachliches wie auch ein gelenktes „Interview" abzuliefern hatte, errechnet sich die Anzahl der zu führenden Interviews mit 64*2 = 128.

Zur Bezeichnung der Informanten und Interviews ist es geschickt, mit geeigneten Kodierungen zu arbeiten. Im AsiCa-Korpus ist jeder Herkunftsort mit einer drei Buchstaben langen Abkürzung bezeichnet, die Generation wird durch 1|2 abgekürzt, das Geschlecht durch m|w und das Merkmal „Migrant" durch I|D (I für Italien, D für Deutschland). Zur Illustration hier die Sigle des männlichen Migranten der zweiten Generation aus Luzzi:

Luz-2-m-D

Da sämtliche Merkmalsausprägungen stets mit einer identischen Anzahl von Zeichen kodiert sind, ist das Setzen der Trennstriche überflüssig. Man kann also ohne jeglichen Informationsverlust schreiben: Luz2mD

Die vollständige Kombination all dieser Merkmalsausprägungen ergibt folgende Informanten-Liste:[6]

[6] Siglen zwischen runden Klammern stehen für Informanten, die in der Realität nicht gefunden werden konnten. Entsprechendes Material ist demnach nicht im Korpus enthalten.

Luz1mI Luz1mD Luz1wI Luz1wD Luz2mI Luz2mD Luz2wI Luz2wD Car1mI Car1mD Car1wI Car1wD Car2mI Car2mD Car2wI Car2wD Bel1mI Bel1mD Bel1wI Bel1wD Bel2mI Bel2mD Bel2wI Bel2wD Spi1mI Spi1mD Spi1wI Spi1wD Spi2mI Spi2mD Spi2wI Spi2wD Mil1mI Mil1mD Mil1wI Mil1wD Mil2mI Mil2mD Mil2wI Mil2wD Biv1mI Biv1mD Biv1wI Biv1wD Biv2mI Biv2mD Biv2wI Biv2wD Pol1mI Pol1mD Pol1wI (Pol1wD) Pol2mI (Pol2mD) Pol2wI (Pol2wD) Bag1mI Bag1mD Bag1wI Bag1wD (Bag2mI) Bag2mD (Bag2wI) Bag2wD

Für die Bezeichnung der einzelnen Interviews muss dann noch je Informant zwischen gelenktem und spontansprachlichem Interview unterschieden werden. Auch dies erfolgt mittels Kodierung, wobei für die Spontansprache die Sigle „Q“ und für die gelenkten Interviews die Sigle „D“ gewählt wurden. Die beiden Interviews des oben zitierten Informanten aus Luzzi sind also durch Luz2mDQ und Luz2mDD bezeichnet.[7]

Ein wesentlicher Vorteil des vorgestellten Systems besteht darin, dass Informanten und Interviews sehr bequem bezüglich einzelner oder auch mehrer Merkmale gefiltert werden können. So lassen sich durch die Verwendung von Platzhaltern Merkmalsgruppen extrahieren. Gesetzt den Fall, ein Fragezeichen (?) stünde für exakt ein beliebiges Zeichen, so würde die folgende Zeichenfolge sämtliche spontansprachlichen Interviews von Frauen der ersten Generation bezeichnen: ???1w?D

Erst nach Festlegung des vorgestellten Schemas wurde mit der eigentlichen Datenerhebung begonnen, d.h. wurden mit dem ausgewählten Personenkreis Interviews durchgeführt. Die dabei entstandenen digitalen Audioaufnahmen erhielten Dateinamen, die sich wiederum an dem skizzierten Schema orientierten (z.B. Mil2mDQ1.wav bzw. nach Konvertierung Mil2mDQ1.mp3). Für den Einsatz einer relationalen ↗Datenbank war schließlich noch die, in diesem Fall phonetische, Transkription der Audiodateien erforderlich. Dieser Schritt erfolgte unter Verwendung des Programms „Praat“ (s. unten S. 107), wobei die dabei entstandenen Textdateien wiederum Namen erhielten, die sich am festgelegten Schema orientierten (z.B. Mil2mDQ1.TextGrid). Da relationale ↗Datenbanken grundsätzlich ↗Tabellengestalt des zu verarbeiteten Materials erfordern, mussten die Textgrid-Dateien des Programms Praat noch mit einem speziell für diesen Zweck geschriebenen kleinen Com-

[7] Im AsiCa-Korpus folgen hinter der Kodierung der Interview-Gattung noch weitere Kodierungen, auf die hier nicht eingegangen werden muß.

puterprogramm, einem AWK-Skript, entsprechend transformiert werden.

Ausschnitt aus dem Praat-Textgrid-File Mil2mDQ1.TextGrid:

```
...
   intervals [4]:
     xmin = 10.537481977023612
     xmax = 14.798985674640702
     text = "mmo # in dialettu militis o paravatotu o
     miskatu kû mil kû missinisi"
...
```

Gestalt nach Umwandlung mittels eines AWK-Skripts (zur besseren Lesbarkeit in eine Word-↗Tabelle umgewandelt):

dateiname	intervall	position	xmin	xmax	Token
Mil2mDQ1	4	1	10.537	14.799	mmo
Mil2mDQ1	4	2	10.537	14.799	#
Mil2mDQ1	4	3	10.537	14.799	in
Mil2mDQ1	4	4	10.537	14.799	dialettu
Mil2mDQ1	4	5	10.537	14.799	militis
Mil2mDQ1	4	6	10.537	14.799	o
Mil2mDQ1	4	7	10.537	14.799	paravatotu
Mil2mDQ1	4	8	10.537	14.799	o
Mil2mDQ1	4	9	10.537	14.799	miskatu
Mil2mDQ1	4	10	10.537	14.799	kû
Mil2mDQ1	4	11	10.537	14.799	mil
Mil2mDQ1	4	12	10.537	14.799	kû
Mil2mDQ1	4	13	10.537	14.799	missinisi

Die folgende Illustration zeigt die ↗Tabelle nach dem Import in eine MySQL-↗Datenbank:[8]

[8] Die dargestellte Oberfläche stammt von dem Datenbankverwaltungstool "phpMyAdmin" (s. unten S. 147). Die Abweichungen in den Spalten xmin und xmax sind Rundungen geschuldet.

interview	intervall	position	xmin	xmax	wort
Mil2mDQ1	4	1	10.540	14.800	mmo
Mil2mDQ1	4	2	10.540	14.800	#
Mil2mDQ1	4	3	10.540	14.800	in
Mil2mDQ1	4	4	10.540	14.800	dialettu
Mil2mDQ1	4	5	10.540	14.800	militis
Mil2mDQ1	4	6	10.540	14.800	o
Mil2mDQ1	4	7	10.540	14.800	paravatotu
Mil2mDQ1	4	8	10.540	14.800	o
Mil2mDQ1	4	9	10.540	14.800	miskatu
Mil2mDQ1	4	10	10.540	14.800	kû
Mil2mDQ1	4	11	10.540	14.800	mil
Mil2mDQ1	4	12	10.540	14.800	kû
Mil2mDQ1	4	13	10.540	14.800	missinisi

Abbildung 3: Tokenisiertes Praat-Intervall nach Import in eine MySQL-↗Datenbank

Der entscheidende Mehrwert, der durch den Import in eine relationale ↗Datenbank erzielt wird, besteht darin, dass das primäre Sprachmaterial der Datenerhebung durch die Anlagerung von ↗Metadaten erweitert werden kann. Diese ↗Metadaten können ganz unterschiedlicher Natur sein, und es ist nicht ausgeschlossen, dass ein einmal zu einem bestimmten Zweck erhobenes Material zu einem späteren Zeitpunkt mit ↗Metadaten verknüpft wird, an die bei der Erhebung gar nicht gedacht worden war. Durch die Anlagerung von ↗Metadaten entsteht überdies die Möglichkeit, Beziehungen zwischen unterschiedlichen ↗Metadaten herzustellen und diese ihrerseits zu analysieren. All dies empfiehlt den Einsatz einer relationalen ↗Datenbank.

Eine im vorliegenden Fall sehr naheliegende Erweiterung des Datenbestandes um ↗Metadaten besteht in der morphosyntaktischen ↗Etikettierung des Materials.[9] Die simpelste Methode, dies zu tun, besteht in der Erweiterung der bestehenden ↗Tabelle um zusätzliche Spalten. Die Registrierung der Wortart könnte also folgendermaßen aussehen:

[9] Neben "↗Etikettierung" wird auch "↗Tagging" als Synonym für die Verknüpfung von ↗Metadaten mit Primärdaten verwendet.

interview	intervall	position	xmin	xmax	wort	wortart
Mil2mDQ1	4	1	10.540	14.800	mmo	AVtem
Mil2mDQ1	4	2	10.540	14.800	#	INCOMPR
Mil2mDQ1	4	3	10.540	14.800	in	Prep
Mil2mDQ1	4	4	10.540	14.800	dialettu	N
Mil2mDQ1	4	5	10.540	14.800	militis	AG
Mil2mDQ1	4	6	10.540	14.800	o	CCo
Mil2mDQ1	4	7	10.540	14.800	paravatotu	AG
Mil2mDQ1	4	8	10.540	14.800	o	CCo
Mil2mDQ1	4	9	10.540	14.800	miskatu	PPP
Mil2mDQ1	4	10	10.540	14.800	kû	v
Mil2mDQ1	4	11	10.540	14.800	mil	v
Mil2mDQ1	4	12	10.540	14.800	kû	v
Mil2mDQ1	4	13	10.540	14.800	missinisi	N

Abbildung 4: Tokenisiertes Praat-Intervall, erweitert um Wortarten-↗Etikettierung

Es versteht sich von selbst, dass im Prinzip eine beliebige Anzahl weiterer Kolumnen angefügt und auf diese Weise nahezu unbegrenzt weitere ↗Metadaten angelagert werden können. Zur Vermeidung von Redundanz und gleichzeitigen Sicherung von Datenkonsistenz und –integrität werden inhaltlich zusammengehörige ↗Metadaten häufig in eigene ↗Tabellen ausgelagert und wird ihr Bezug zu den Primärdaten durch die Verknüpfung mittels Identifikationsnummern hergestellt. Man bezeichnet dieses Verfahren als ↗„Normalisierung“.

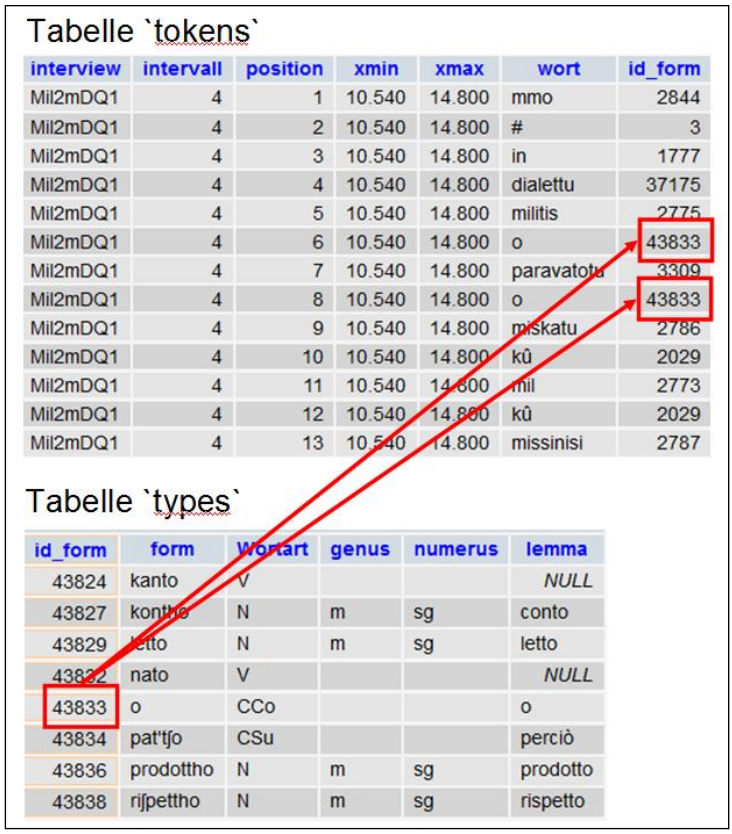


Tabelle `tokens`

interview	intervall	position	xmin	xmax	wort	id_form
Mil2mDQ1	4	1	10.540	14.800	mmo	2844
Mil2mDQ1	4	2	10.540	14.800	#	3
Mil2mDQ1	4	3	10.540	14.800	in	1777
Mil2mDQ1	4	4	10.540	14.800	dialettu	37175
Mil2mDQ1	4	5	10.540	14.800	militis	2775
Mil2mDQ1	4	6	10.540	14.800	o	43833
Mil2mDQ1	4	7	10.540	14.800	paravatotu	3309
Mil2mDQ1	4	8	10.540	14.800	o	43833
Mil2mDQ1	4	9	10.540	14.800	miskatu	2786
Mil2mDQ1	4	10	10.540	14.800	kù	2029
Mil2mDQ1	4	11	10.540	14.800	mil	2773
Mil2mDQ1	4	12	10.540	14.800	kù	2029
Mil2mDQ1	4	13	10.540	14.800	missinisi	2787

Tabelle `types`

id_form	form	Wortart	genus	numerus	lemma
43824	kanto	V			*NULL*
43827	kontho	N	m	sg	conto
43829	letto	N	m	sg	letto
43832	nato	V			*NULL*
43833	o	CCo			o
43834	pat'tʃo	CSu			perciò
43836	prodottho	N	m	sg	prodotto
43838	riʃpettho	N	m	sg	rispetto

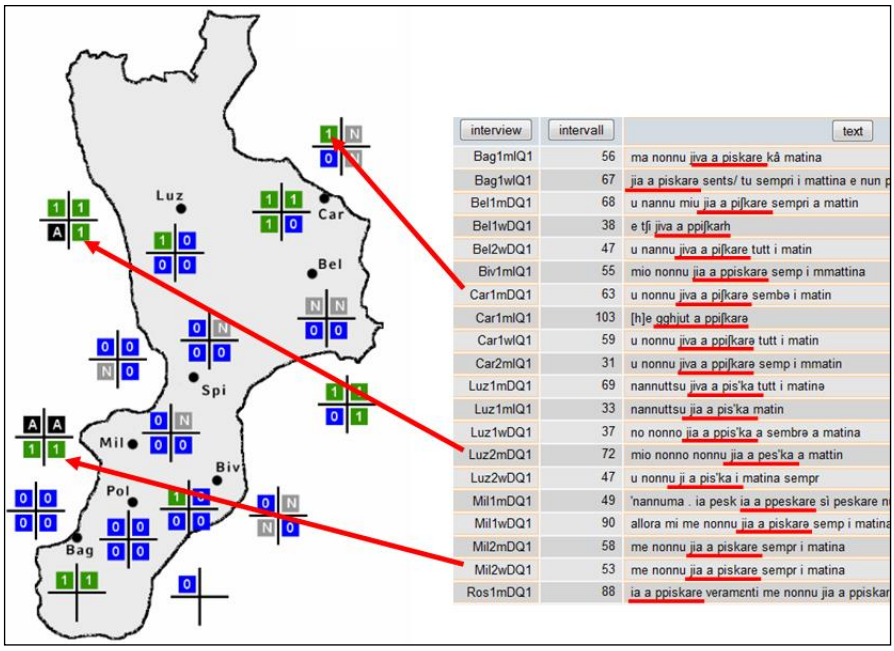
Abbildung 5: ↗„Normalisierung“ – Auslagerung morphosyntaktischer Etikettierungsdaten in eine eigene Datenbanktabelle

Die ↗Datenbank erlaubt schließlich die Analyse des kompletten Datenbestandes in der Weise, dass Primär- und ↗Metadaten nahezu beliebig miteinander kombiniert werden können. In relationalen Datenbanksystemen wie z.B. MySQL kommt dabei die spezielle Abfragesprache ↗SQL ("Structured Query Language") zum Einsatz. Die zumindest ansatzweise Beherrschung dieser Sprache ist Voraussetzung für den effizienten Einsatz einer relationalen ↗Datenbank. Bis zu einem gewissen Grad kann man sich auf die Datenbankverwaltungsoberfläche PhpMyAdmin stützen, die die wichtigsten Grundfunktionen für die Verwaltung und Analyse der Daten in der ↗Datenbank bequem über eine Weboberfläche zugänglich macht.

Im folgenden Beispiel wird nach Informanten gesucht, die den Stimulus „Mio nonno andava a pescare sempre di mattina“ (Stimulus F14) unter Verwendung eines Infinitivs wiedergegeben haben:

```
select
     text.Interview interview,
     text.intervall,
     text.text
from text
left JOIN wort wort_1
  ON text.Interview = wort_1.Interview
     AND text.Text_nr = wort_1.Text_nr
left JOIN formen form_1 ON wort_1.id_form =
     form_1.id_form
left JOIN lemmata lemma_1 ON form_1.id_lemma =
     lemma_1.id
left JOIN wort wort_2 ON text.Interview =
     wort_2.Interview
     AND text.Text_nr = wort_2.Text_nr
left JOIN formen form_2 ON wort_2.id_form =
     form_2.id_form
left JOIN lemmata lemma_2 ON form_2.id_lemma =
     lemma_2.id
where
     text.sprecher NOT like 'E%'
     AND text.Interview LIKE '%Q1'
     AND (
          lemma_1.lemma = 'ire'
          OR lemma 1.lemma = 'andare'
          OR lemma_1.lemma = 'vadere'
     )
     AND (
          form_2.modus LIKE 'inf'
          OR form_2.modus LIKE 'pinf'
          )
     AND (
          wort_2.id_wort - wort_1.id_wort <=2
          AND wort_2.id_wort - wort_1.id_wort >0
     )
     AND text.quest like 'F14:%'
group by text.interview
;
```

Das Ergebnis dieser Abfrage präsentiert sich zunächst wieder in Tabellengestalt. Die Verarbeitung dieser Daten mit Hilfe geeigneter Programmiersprachen (im Falle von ASiCa „PHP“), gestattet deren visuelle Aufbereitung etwa in Gestalt einer Landkarte, auf der verschiedene Abfrageergebnisse synoptisch, unterschieden durch Farbgebung in ihrer Verteilung in der Fläche dargestellt werden.



interview	intervall	text
Bag1mlQ1	56	ma nonnu jiva a piskare kâ matina
Bag1wlQ1	67	jia a piskarə sents/ tu sempri i mattina e nun p
Bel1mDQ1	68	u nannu miu jia a piʃkare sempri a mattin
Bel1wDQ1	38	e tʃi jiva a ppiʃkarh
Bel2wDQ1	47	u nannu jiva a piʃkare tutt i matin
Biv1mlQ1	55	mio nonnu jia a ppiskarə semp i mmattina
Car1mDQ1	63	u nonnu jiva a piʃkarə sembə i matin
Car1mlQ1	103	[h]e qqhjut a ppiʃkarə
Car1wlQ1	59	u nonnu jiva a ppiʃkarə tutt i matin
Car2mlQ1	31	u nonnu jiva a ppiʃkarə semp i mmatin
Luz1mDQ1	69	nannuttsu jiva a pis'ka tutt i matinə
Luz1mlQ1	33	nannuttsu jia a pis'ka matin
Luz1wDQ1	37	no nonno jia a ppis'ka a sembrə a matina
Luz2mDQ1	72	mio nonno nonnu jia a pes'ka a mattin
Luz2wDQ1	47	u nonnu ji a pis'ka i matina sempr
Mil1mDQ1	49	'nannuma . ia pesk ia a ppeskare sì peskare n
Mil1wDQ1	90	allora mi me nonnu jia a piskarə semp i matina
Mil2mDQ1	58	me nonnu jia a piskare sempr i matina
Mil2wDQ1	53	me nonnu jia a piskare sempr i matina
Ros1mDQ1	88	ia a ppiskare veramenti me nonnu jia a ppiskar

Abbildung 6: Kartographische Abbildung einer analytischen Datenbankabfrage

Auf der oben abgebildeten Karte stehen je zwei Vierergruppen in der Nähe je eines der Ortspunkte für einen Familienverband, wobei die, die sich außerhalb des Konturs von Kalabrien befinden, die nach Deutschland migrierten Familienmitglieder symbolisieren. Innerhalb jeder Vierergruppe, repräsentieren jeweils die linken Symbole die Männer, die rechten die Frauen. Die obere Reihe steht für die Mitglieder der ersten Generation, die untere für die Mitglieder der zweiten. Eine geeignete Datenstruktur vorausgesetzt, lassen sich in der beschriebenen Weise nahezu beliebige Analysen und Visualisierungen durchführen.

Die Präsentation des AsiCa-Korpus sollte exemplarisch die Möglichkeiten der Datenanalyse und -präsentation vorstellen, die sich bei geschickter Konzeption, geeigneter Datenstrukturierung und durch Einsatz einer relationalen ↗Datenbank ergeben können. Im Folgenden werden nun technische Hilfsmittel und Strategien vorgeführt, die die Aufbereitung von Korpusmaterial zur weiteren atuomatischen Bearbeitung für den Import in eine relationale ↗Datenbank erleichtern bzw. überhaupt erst ermöglichen können. Wir beginnen mit der Beschreibung eines der fundamentalen Werkzeuge, die für die elektronische Bearbeitung von Texten unverzichtbar sind: dem Editor „VIM“.

3 Grundlegende Konzepte, Verfahren und Werkzeuge

3.1 Der Editor VIM

Für die Arbeit mit Texten im Sinne der Informationstechnologie werden reine Text-Editoren benötigt und auf keinen Fall Textverarbeitungsprogramme wie etwa Microsoft Word, die zusätzlich zu Textinformationen auch Formatieranweisungen enthalten. Es gibt zahlreiche solcher Texteditoren (u.a. die auf jedem Windows-Rechner vorinstallierten Programme „Editor“ oder „WordPad“), von denen jedes Vor- und auch Nachteile besitzt.[10]

Ein excellenter Editor, der sich im Rahmen einschlägiger Arbeiten an Textkorpora bestens bewährt hat, ist das Programm „VIM“. VIM hat folgende wesentliche Vorzüge:

- es ist kostenlos
- es ist „Open Source“
- es ist für alle gängigen ↗Betriebssysteme (u.a. Windows, Mac, Linux) verfügbar

Das Programm kann von folgender Webseite heruntergeladen werden:

http://www.vim.org

Die Installation ist vollkommen unproblematisch. Man folgt einfach den Anweisungen auf dem Bildschirm.

Der Umgang[11] mit VIM ist insofern gewöhnungsbedürftig, als die Bedienung von dem in der Windows-Welt Üblichen abweicht. Einer der großen Vorteile von VIM besteht darin, dass es vollständig über die Tastatur gesteuert werden kann. Es gibt zwar auch die Möglichkeit, VIM mit der Maus zu bedienen, wir möchten davon aber abraten, da die Steuerung via Tastatur wesentlich effizienter ist und ein Teil der Editierbefeh-

[10] Neben dem hier vorgestellten Editor VIM sind auch die Programme Notepad++ oder Sublime Text nach unserer Auffassung empfehlenswert. Eine (sicher nicht vollständige) Liste dieser und weiterer Editoren findet sich in Wikipedia.

[11] Hilfreiche Informationen und Erläuterungen finden sich u.a. unter folgenden Adressen: http://de.wikipedia.org/wiki/Vim; https://de.wikibooks.org/wiki/Vi-Befehlsreferenz:_%C3%9Cbersicht

le auch automatisiert verwendet werden kann (siehe z.B. den Streameditor sed).

Nach der Installation liegt das Programm in zwei Versionen vor: einer, die von der Kommandozeile aus aufgerufen wird (vim.exe), und einer, die mit einer graphischen Benutzeroberfläche wie ein übliches Windows-Programm in einem eigenen Fenster erscheint (gvim.exe). Im Folgenden ist nur von letzterer die Rede.

Das Programm wird durch Doppelklick auf das entsprechende Symbol gestartet. In der Grundeinstellung erscheint dann ein Fenster mit einem weißen Hintergrund und dunkler Schrift. Besonders praktisch ist, dass bei der Installation von VIM die Kontextmenüs des Windows-Explorers um einen Eintrag „Editiere mit Vim“ ergänzt werden. Auf diese Weise lassen sich beliebige Dateien öffnen, auch z.B. binäre Programmdateien (*.exe) – was allerdings wenig Sinn macht. Interessant aber ist, auf diese Weise einmal eine Word-Datei zu öffnen. Man kann dann deren „wahre“ Gestalt sehen und erkennt, dass diese weit mehr Informationen enthalten als den blanken Text, der in der Datei abgelegt ist. Dies ist auch der Grund, warum sich Word eben gerade *nicht* für die Arbeit im Sinne der Korpuslinguistik eignet.

Wir unterscheiden vier verschiedene VIM-Funktionsmodi:

- Befehlsmodus
- Eingabemodus
- Kommandozeilenmodus
- Visual-Modus

Wir gehen nun davon aus, dass man eine reine Textdatei mit gvim geöffnet hat. Nach dem Öffnen dieser Datei befindet sich VIM im **Befehls-Modus**, erkennbar an der Blockform des ↗Cursors:

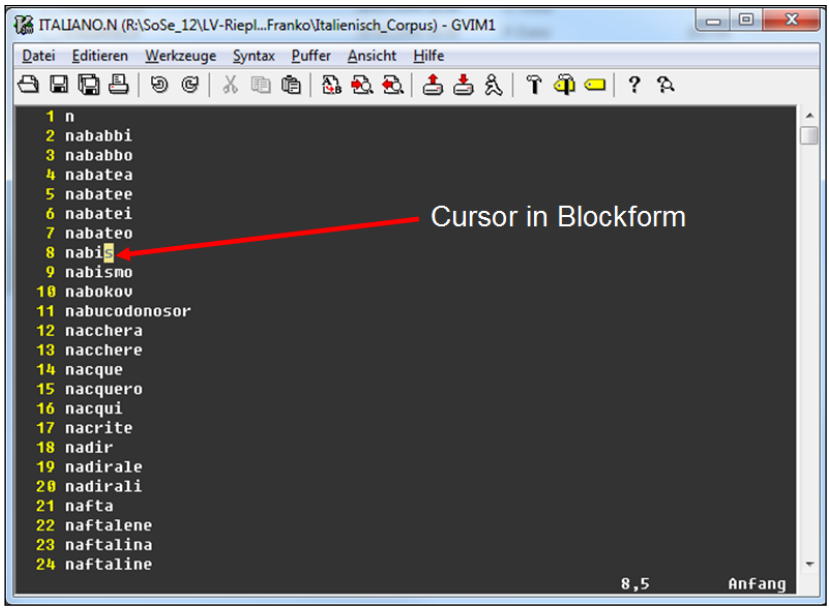


Abbildung 7: Der Editor VIM im Befehls-Modus

Im **Befehlsmodus** ist es nicht möglich, Text einzugeben. Nahezu jede Taste auf der Tastatur besitzt eine spezielle Funktion. Wir beschränken uns hier auf die wichtigsten der zur Verfügung stehenden Befehle (VIM unterscheidet zwischen Groß- und Kleinbuchstaben!):

Taste	Funktion
h	↗Cursor nach links
j	↗Cursor nach unten
k	↗Cursor nach oben
l	↗Cursor nach rechts
gg	↗Cursor an den Anfang der Datei
G	↗Cursor an das Ende der Datei
u	letzte Aktion rückgängig machen („undo“)
y[Adresse]	Inhalt von „Adresse“ in den Zwischenspeicher kopieren (yy kopiert die ganze ↗Zeile). Adresse kann auch sein: w (Wort)
p	Den Inhalt der Zwischenablage an der Cursorposition einfügen
i	Wechsel in den Einfüge-Modus. Einfügemarke springt *vor* („insert“) die aktuelle Cursorposition
I	Wechsel in den Einfüge-Modus. Einfügemarke springt an den Anfang der aktuellen ↗Zeile

a	Wechsel in den Einfüge-Modus. Einfügemarke springt *hinter* („append“) die aktuelle Cursorposition
A	Wechsel in den Einfüge-Modus. Einfügemarke springt an das Ende der aktuellen ↗Zeile
o	Wechsel in den Einfüge-Modus. Erzeugung einer neuen ↗Zeile *unterhalb* der aktuellen, Einfügemarke springt in die neue ↗Zeile
O	Wechsel in den Einfüge-Modus. Erzeugung einer neuen ↗Zeile *oberhalb* der aktuellen, Einfügemarke springt in die neue ↗Zeile
/	↗Cursor springt an den unteren Fensterrand und wartet auf die Eingabe eines Suchmusters, wobei auch Reguläre Ausdrücke erlaubt sind! Nach Drücken der Enter-Taste springt der ↗Cursor zum ersten Vorkommen des gesuchten ↗Strings.
*	VIM markiert alle Vorkommen des Wortes bzw. der Zeichenkette, auf dem der ↗Cursor gerade steht. Durch Drücken der Taste „n“ springt der ↗Cursor zum jeweils nächsten Vorkommen des Wortes/↗Strings
n	↗Cursor springt zum jeweils nächsten Vorkommen des aktuell definierten Suchstrings
ga	ermittelt den Zahlenwert des Zeichens, auf dem sich der ↗Cursor gerade befindet und schreibt ihn an den linken unteren Bildschirmrand (steht z.B. der ↗Cursor auf dem Zeichen ʁ [‚latin letter small capital inverted r‘], und man drückt die Tastenfolge ‚ga‘, so schreibt VIM an die beschriebene Stelle den Zahlenwert dez641 (zusätzlich gibt VIM den hexadezimalen [x0281] und den oktalen [oct1201] Zahlenwert aus).
.	wiederholt den zuletzt aufgerufenen Befehl
:	Wechsel in den Kommandozeilen-Modus
strg + q + u	Erlaubt die Eingabe eines Zeichens mittels ↗Unicode-Zahlenwerten. Nach Drücken des „u“ müssen exakt 4 Ziffern eingegeben werden. Nach Eingabe der vierten Ziffer erscheint das entsprechende Zeichen an der Stelle der Einfügemarke (ggf. in Gestalt eines kleinen Rechtecks, wenn der Editor über keine passende ↗Glyphe verfügt). Diese Methode ist immer dann erforderlich, wenn Zeichen nicht direkt über die Tastatur eingeben werden können, in der Regel sind dies Zeichen oberhalb des ↗ASCII-/↗Unicode-Zahlenwertes 255.

strg + q + U	Zeichen jenseits des ↗Unicode-Zahlenwertes xFFFF können in o.a. Befehlssequenz durch Verwendung des Großbuchstabens „U“ eingegeben werden. Nach Drücken des „U“ müssen exakt 8 Ziffern eingegeben werden. Nach Eingabe der achten Ziffer erscheint das entsprechende Zeichen an der Stelle der Einfügemarke.

Durch Drücken der Tasten i, I, a, A, o oder O wechselt VIM in den **Einfügemodus**. Nunmehr erscheint der ↗Cursor als senkrechter Strich (sog. Einfügemarke), außerdem steht am unteren Fensterrand der Hinweis „EINFÜGEN“:

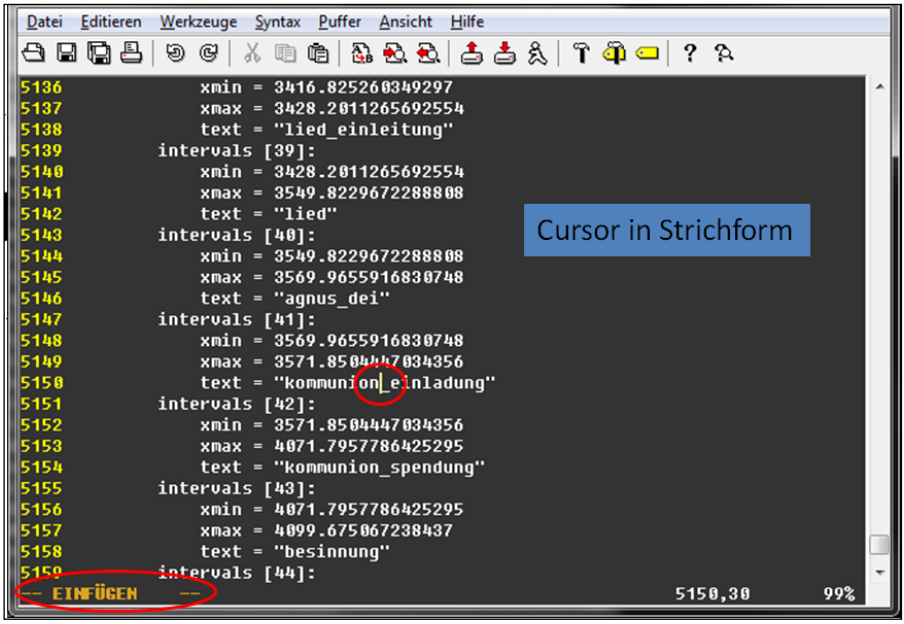


Abbildung 8: Der Editor VIM im Einfüge-Modus

Nur in diesem Modus ist die Eingabe von Text über die Tastatur möglich.

Das Drücken der Taste „:“ (Doppelpunkt) bewirkt den Wechsel in den **Kommandozeilen-Modus**. Der ↗Cursor springt an den linken unteren Bildschirmrand und wartet auf die Eingabe von Befehlen. Die nun zur Verfügung stehenden Optionen sind nahezu unbegrenzt. Wir konzentrieren uns hier wiederum auf diejenigen, die erfahrungsgemäß im Rahmen der Korpuslinguistik ständig benötigt werden.

Befehl	Funktion
:w	Speichert die gerade geöffnete Datei ab („write“)
:q	Schließt die geöffnete Datei und beendet VIM („quit“)
:wq	Speichert die Datei und beendet VIM
:q!	Schließt die Datei *ohne* zu speichern!
:se	Zeigt die aktuellen Einstellungen an („settings“)
:se ff=unix	Legt das Dateiformat auf Unix fest (⇒ Konsequenz für die Markierung des Zeilenendes; s. unten S. 55); analog: se ff=dos. Der Aufruf des Kommandos ohne das =-Zeichen und den nachfolgenden Wert zeigt die aktuell gültige Einstellung an. Dies gilt analog für alle mit :se ansprechbaren Parameter.
:se encoding=utf-8	Legt die in der geöffneten Datei verwendete Kodierung fest (hier ↗UTF-8).
:se fileencoding=utf-8	Legt die Kodierung der Datei beim Schreiben auf das Speichermedium auf ↗UTF-8 fest; analog: se fileencoding=latin1
:se bomb	Beim Speichern der Datei wird an deren Anfang ein Byte Order Mark (↗BOM) geschrieben.
:se nobomb	Beim Speichern der Datei wird kein ↗BOM an den Anfang der Datei geschrieben. Sollte die Datei ursprünglich ein ↗BOM besessen haben, so wird dieses entfernt.
:se nu	Schaltet die Zeilennumerierung ein
:se nonu	Schaltet die Zeilennumerierung aus
:nohl	Löscht die farbliche Hervorhebung (highlighting) von zuvor gesuchten Wörtern/Zeichenketten
:[Bereich]s/Suchstring/Ersetzungsstring/gc	Ersetzen von Suchstring durch Ersetzungsstring im angegebenen Be-

	reich. Der Bereich wird durch Zeilennummern angegeben, , % steht für die komplette Datei. Ohne Bereichsangabe erfolgt die Ersetzung nur in der aktuellen ↗Zeile. Das „g“ hinter dem letzten Schrägstrich bedeutet, dass in jeder ↗Zeile nicht nur der jeweils *erste* Treffer, sondern alle Vorkommen des Suchstrings ersetzt werden sollen. Das nachfolgende „c“ steht für „confirm“ und bedeutet, dass jede einzelne Ersetzung bestätigt werden muss. Der Suchstring kann auch aus einem Regulären Ausdruck bestehen.[12]

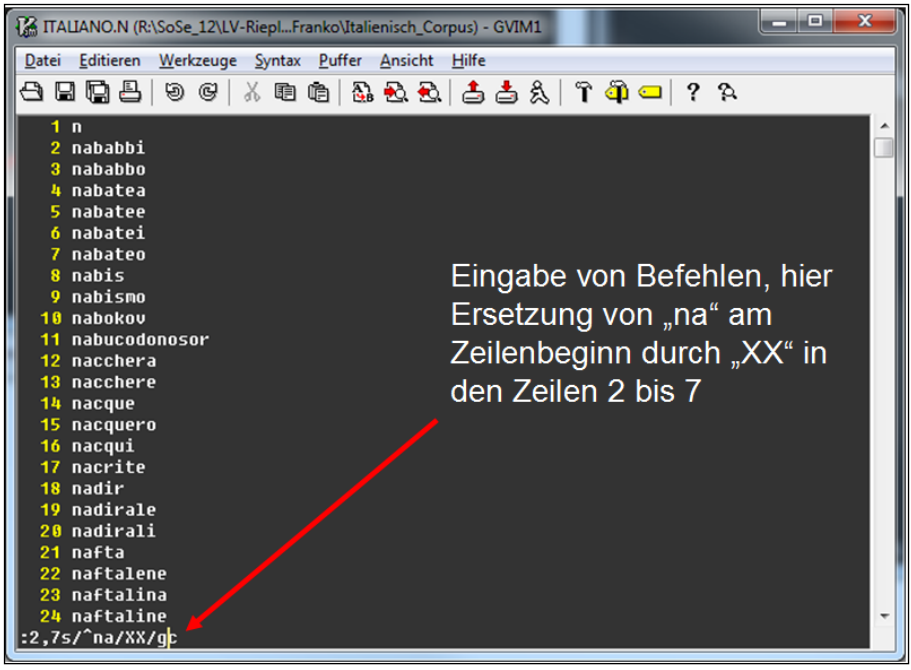


Abbildung 9: Der Editor VIM im Kommandozeilen-Modus

[12] Der Slash als Begrenzer des Such- bzw. Ersetzungsstrings kann durch andere Zeichen (z.B. "=") ersetzt werden. Dies ist besonders dann von Vorteil, wenn der Slash im Such- bzw. Ersetzungsstring vorkommt: :[Bereich]s=Suchstring=Ersetzungsstring=gc

Schließlich existiert noch der sog. „**Visual-Modus**“, der dadurch aktiviert wird, dass man mit der Maus oder der Tastatur größere Textpassagen markiert, die sodann ausgeschnitten, kopiert oder auch gelöscht werden können:

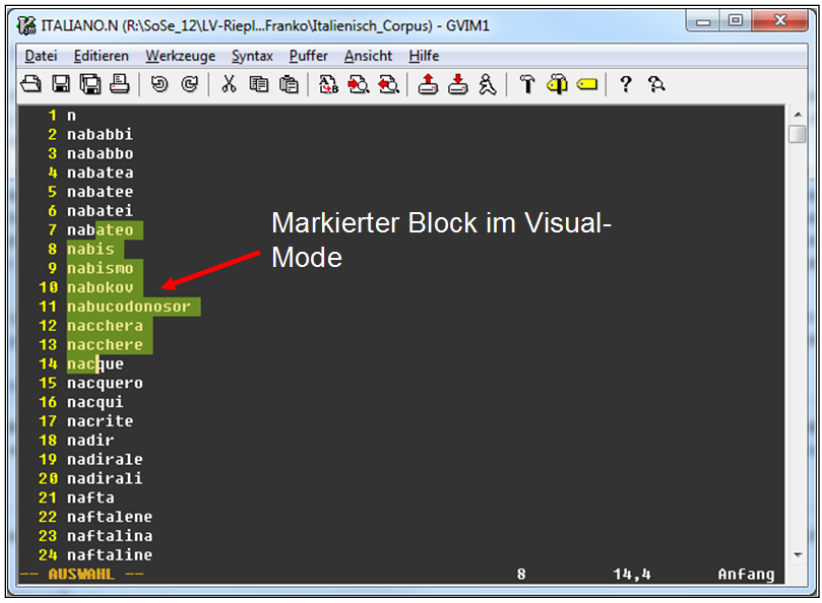


Abbildung 10: Der Editor VIM im Visual-Mode

VIM besitzt eine hervorragende eingebaute **Hilfefunktion**. Man erreicht sie über den Kommandozeilen-Modus mit dem Befehl „:he“. VIM zeigt dann die Titelseite der Hilfe. Innerhalb der Hilfe bewegt man sich wie auch sonst in VIM üblich (alle Befehle – so auch die Suchfunktion „/“ – stehen zur Verfügung; die Hilfe-Dateien sind allerdings schreibgeschützt, können also nicht verändert werden). Ein Doppelklick auf die farblich hervorgehobenen Links öffnet die betreffenden Dateien und/oder bewirkt einen Sprung an die entsprechende Stelle des Hilfetextes. Gibt man „:he“ gefolgt von einem speziellen VIM-Befehl ein, erhält man gezielt Hilfe zu diesem Befehl. Die Hilfefunktion wird wie jede andere Datei auch mit :q verlassen.

Sowohl der Kommandozeilen- wie auch der Einfüge-Modus werden durch **Drücken der Taste „ESC“** wieder verlassen. VIM wechselt dann wieder in den Befehlsmodus.

3.2 Zeichenkodierung

3.2.1 Grundsätzliches

Bei der Erfassung von elektronischen Texten und ihrer Ablage in Dateien ist unbedingt auf die Zeichenkodierung zu achten. Diese muss – ungeachtet möglicherweise unterschiedlicher Herkunft der Texte – einheitlich sein. Zum Verständnis: **Computer kennen keine Buchstaben, sondern nur Zahlen**. Eine einheitliche Kodierung ist dann gegeben, wenn grundsätzlich einem Buchstaben stets ein und dieselbe Zahl zugeordnet ist. Beispiel:

In Windows entspricht der deutsche Umlaut „ü“ dem Zahlenwert dez252:

ü ⇒ dez252

Unter Mac OS 9 entspricht der deutsche Umlaut „ü“ dem Zahlenwert dez159:

ü ⇒ dez159

Wenn nun ein Teil des gesammelten Materials aus einer Quelle stammt, die die Windows-Kodierung verwendet, ein anderer Teil aber aus einer Quelle, die die Mac-Kodierung verwendet, entsteht Verwirrung:

Es grünt so grün, wenn Spaniens BlŸten blŸh'n.

Der grün unterlegte Abschnitt stammt von einem Macintosh-Computer, der für das „ü“ die Zahl dez159 geschrieben hat, wohingegen der Windows-Rechner an dieser Stelle die Zahl dez252 geschrieben hat (blau unterlegt). Der Zahlenwert dez159 wird vom Windows-↗Betriebssystem als Ÿ interpretiert. Würde man umgekehrt obiges Beispiel auf einem älteren Macintosh-Computer öffnen, sähe das Ergebnis so aus:

Es gr¸nt so gr¸n, wenn Spaniens Blüten blüh'n.

Der Zahlenwert dez252, der von Windows an die Stelle eines „ü“ geschrieben wurde, wird vom Macintosh-↗Betriebssystem als ¸ (Cedilla) interpretiert.

Das Hauptproblem im gegebenen Beispiel besteht in der Uneinheitlichkeit der Kodierung. Sicherlich sind derlei Probleme lösbar, gerade bei größeren Textmengen ist das Verfahren jedoch aufwendig und fehleranfällig.

Achten Sie daher von Anbeginn auf die korrekte und einheitliche Kodierung Ihres Materials!

Verwenden Sie Kodierungsverfahren, die betriebssystemunabhängig sind. Dabei bieten sich zwei Optionen an:

- Kodierung nach dem „↗Betacode"-Verfahren (von uns so genannt nach dem Prototypen, der elektronischen Erfassung altgriechischer Text im „Thesaurus Linguae Greacae" [TLG; http://stephanus.tlg.uci.edu/]).
- Kodierung nach der ↗Unicode-↗Tabelle[13] (technisches Verfahren: ↗UTF-8)

Eine verbindliche Empfehlung, welches Kodierungsverfahren zu verwenden ist, läßt sich nicht geben. Die „richtige" Entscheidung hängt stets vom zu verarbeitenden Material bzw. den verfolgten Analyse-Zielen ab. Besondere Vorsicht ist geboten, wenn Sie Sprachmaterial aus unterschiedlichen elektronischen Quellen in Ihrem Korpus zusammenführen wollen. Es ist davon auszugehen, dass Materialien aus unterschiedlichen Quellen in voneinander abweichenden Kodierungen vorliegen.

3.2.2 Zahlensysteme

Wie bereits gesagt, können Computer ausschließlich Zahlen verarbeiten. Und da die in Computern verwendeten Speicherelemente überdies lediglich zwei Zustände kennen, nämlich „Vorhandensein von Stromspannung" und „Abwesenheit von Stromspannung", können Computer nur mit dem sog. binären Zahlensystem rechnen, das mit den beiden Ziffern 0 und 1 auskommt. Zum besseren Verständnis sei an dieser Stelle grundsätzlich die Funktionsweise von Zahlensystemen erläutert.

[13] Eine datenbankgestützte online-Referenz des ↗Unicode-Standards ist konsultierbar unter https://pma.gwi.uni-muenchen.de:8888/index.php?db=LIPP_unicode&server=8 Benutzername: unicodeITG; Passwort: unicodeITG; über "SQL" ⇨ "Gespeicherte SQL-Abfrage" ist der Zeichenbestand nach Blöcken abrufbar).

Es gibt generell zwei Typen von Zahlensystemen: Additionssysteme (z.B. das römische Zahlensystem oder eine simple Strichliste) und Stellenwertsysteme. Im Folgenden ist nur von Stellenwertsystemen die Rede.

Jedes Stellenwertsystem basiert im Wesentlichen auf vier Grundbegriffen, durch deren Kombination Zahlen dargestellt werden können. Die vier Grundbegriffe lauten:

- Basiszahl
- Position
- Ziffernwert
- Potenz

Der Wert der Basis bestimmt die Anzahl der in einem Zahlensystem benötigten Ziffern. Zur Vermeidung von Mißverständnissen wird in Kontexten, in denen das einer Zahl zugrundeliegende Zahlensystem nicht zweifelsfrei feststellbar ist, folgender Konvention gefolgt: hexadezimalen Zahlen wird ein „x“ vorangestellt (z.B. x11), dezimalen ein „dez“ (dez11), oktalen ein „oct“ und binären ein „bin“.

- Das dezimale Zahlensystem (Basiszahl = 10)

Ziffern (insgesamt 10): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Position	3	2	1	0
Ziffernwert	1	2	5	2
Positionswert=Basis^{Position} * Ziffernwert	=10^{3}*1	=10^{2}*2	=10^{1}*5	=10^{0}*2
Ergebnis	1000	200	50	2

Der Zahlenwert errechnet sich nun aus der Addition der Positionswerte:

1000 + 200 + 50 + 2 = 1252

Sofern das verwendete Medium nur Ja/Nein-Informationen übermitteln kann, ergibt sich ein neues Problem der Kodierung: Die Dezimalzahlen müssen in das duale Zahlensystem, das nur mit Nullen und Einsen auskommt, umgewandelt werden.

Analog zum dezimalen Zahlensystem baut sich das Duale (Binäre) Zahlensystem auf, mit dem Unterschied, dass die Basis nicht 10, sondern 2 ist.

- Das binäre (auch: duale) Zahlensystem (Basiszahl = 2)

Dieses Zahlensystem wird, wie gesagt, von Computern verwendet.

Ziffern (insgesamt 2): 0, 1

Die duale Zahl 1101 ergibt sich folgendermaßen:

Position	3	2	1	0
Ziffernwert	1	1	0	1
Positionswert=Basis^{Position} * Ziffernwert	=2^3*1	=2^2*1	=2^1*0	=2^0*1
Ergebnis	8	4	0	1

Der Zahlenwert errechnet sich wiederum aus der Addition der Positionswerte:

8 + 4 + 0 + 1 = 13

- Basis: 16 (hexadezimales Zahlensystem)

Ziffern (insgesamt 16): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Position	3	2	1	0
Ziffernwert	0	0	9	F
Positionswert=Basis^{Position} * Ziffernwert	=16^3*0	=16^2*0	=16^1*9	=16^0*F
Ergebnis	0	0	144	15

Der Zahlenwert errechnet sich aus der Addition der Positionswerte:

144 + 15 = 159

Die häufige Verwendung des Hexadezimalen Zahlensystems in der Informatik hängt hauptsächlich damit zusammen, dass in Computern quasi ausnahmslos jeweils 8 ↗Bit (= 1 ↗Byte) für die Kodierung von Zeichen verwendet werden. Bei Verwendung des hexadezimalen Systems läßt sich jedes ↗Byte mit genau zwei Ziffern darstellen. Die größtmögliche Zahl, die mit einem ↗Byte dargestellt werden kann ist

1111 = xF

11111111 = xFF

3.2.3 Ermittlung vorliegender Zeichenkodierungen

Es gibt verschiedene Möglichkeiten, die aktuelle Zeichenkodierung eines Textes zu ermitteln. Die Möglichkeiten differieren allein schon in Abhängigkeit vom verwendeten Programm (Browser, Editor, Word). Es ist möglich, dass ein Text, der im einen Programm in einer bestimmten Kodierung vorlag, nach Transfer in ein anderes Programm (z.B. Browser ⇒ VIM) seine Kodierung verändert hat. Dies hängt von der Art und Weise des Transfers ab („speichern unter ...“ versus copy/paste). Es ist nicht möglich, hier alle Fälle zu beschreiben. Es sei hier nur auf dieses Phänomen als ernstzunehmende mögliche Fehlerquelle hingewiesen.

Die Ermittlung vorliegender Kodierungen erfolgt z.B.

- im Browser Firefox (v. 13):

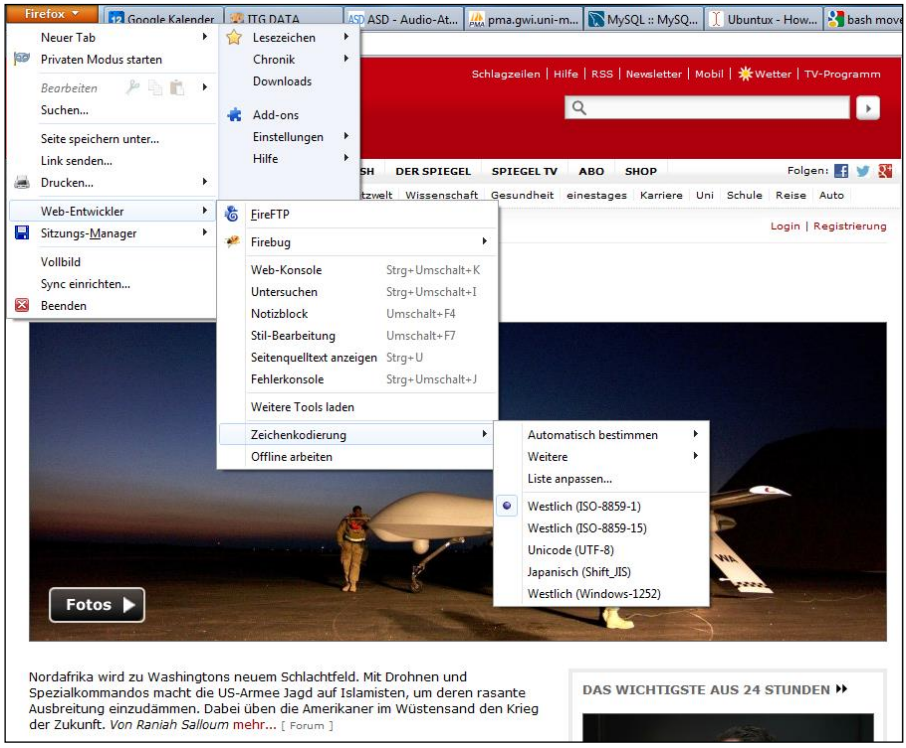


Abbildung 11: Auswahl der zu verwendenden Zeichenkodierung im Browser Firefox (Version 13)

- im Internet-Explorer (nach rechtem Mausklick auf den Seiteninhalt):

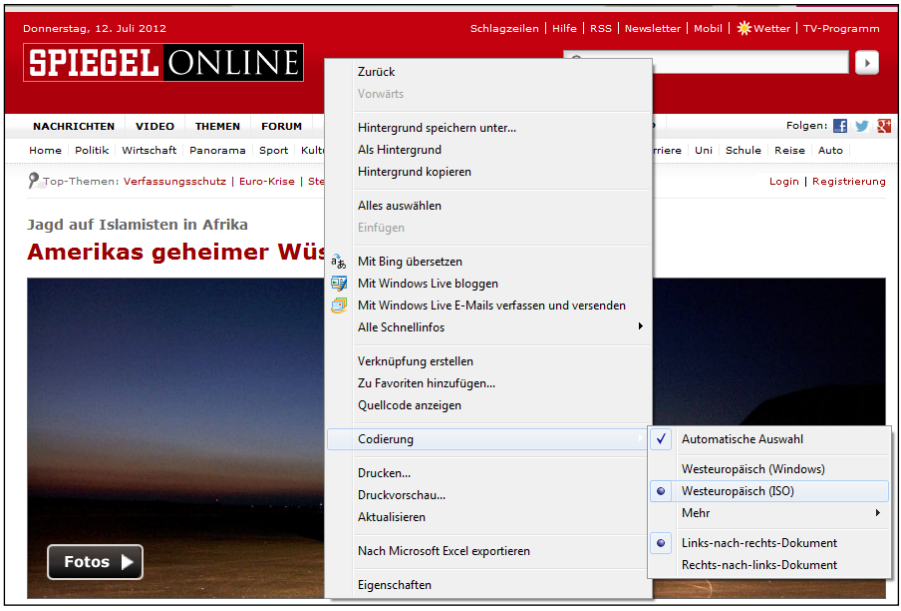


Abbildung 12: Auswahl der zu verwendenden Zeichenkodierung im Browser Internet Explorer (Version 9)

- im Editor VIM durch Eingabe des Kommandos :se fileencoding:

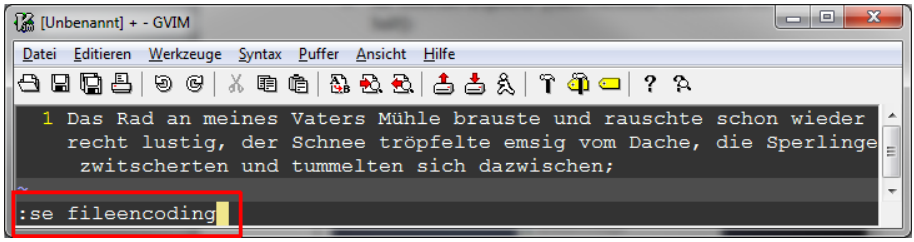


Abbildung 13: Kommando zur Anzeige der aktuell verwendeten Zeichenkodierung im Editor VIM

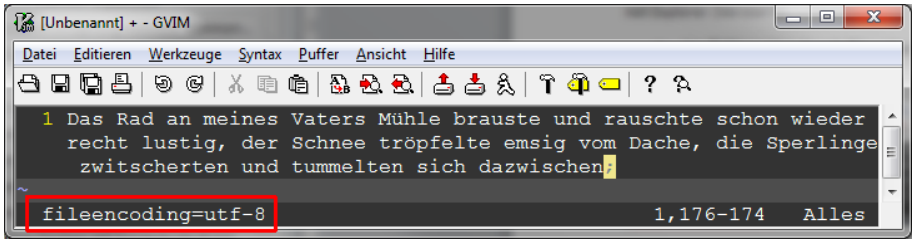


Abbildung 14: Anzeige der aktuell verwendeten Zeichenkodierung im Editor VIM

In den Programmen der Office-Familie ist die Ermittlung der Kodierung häufig nicht leicht ersichtlich. In Word kann die verwendete Kodierung beim Abspeichern des Textes im Format „Nur Text (*.txt)“ definiert werden:

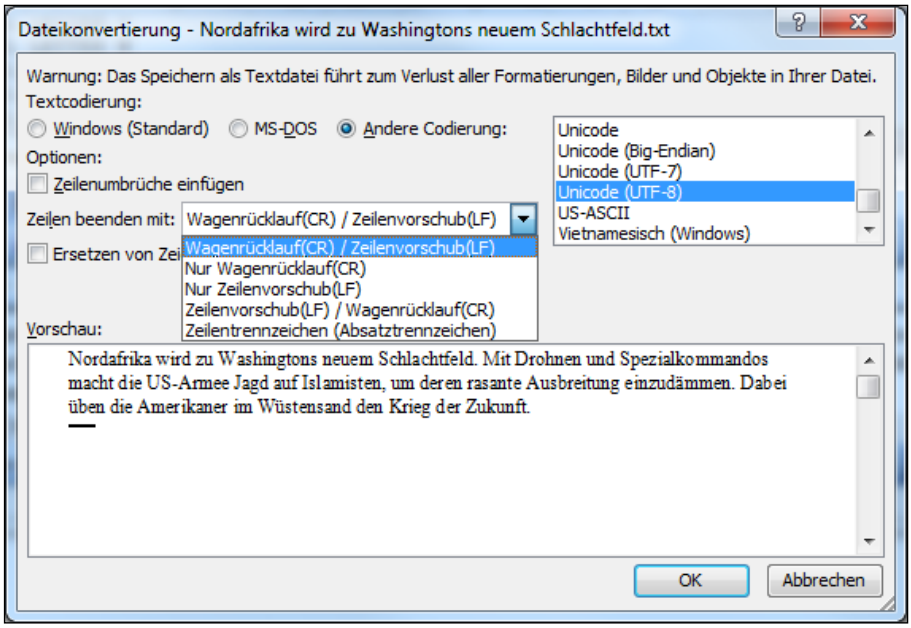


Abbildung 15: Speichern einer Datei in ↗UTF-8-Kodierung mit Microsoft Word bei gleichzeitiger – windows-typischer – Verwendung von CR (= x0d) und LF (= x0a) zur Kodierung des Zeilenendes

Für die Kodierung riskante Operationen sind grundsätzlich alle „Bewegungen“ von Text von einem Programm in ein anderes.[14] Gewöhnen Sie sich an, in diesen Fällen anschließend *immer* die Kodierung zu überprüfen. Nach unseren Erfahrungen ist das Verfahren copy/paste die im Hinblick auf die Kodierung unproblematischste Methode, Text von einem Programm in ein anderes zu bewegen.

Vermeiden Sie unter allen Umständen einen Kodierungs-„Mix“, d.h. das Zusammenführen von Textdaten aus unterschiedlichen Quellen in jeweils unterschiedlicher Kodierung in einer einzigen Datei. Solche Da-

[14] Dies gilt auch für das Versenden von Daten via E-Mail. Bevorzugen Sie in diesem Fall immer die Methode, die Korpusdaten in einem korrekt kodierten und zip-komprimierten Textdokument als Attachment mitzuschicken.

teien sind, sobald sie eine gewisse Größe überschritten haben, de facto nicht mehr zu heilen und damit eigentlich unbrauchbar.

An dieser Stelle können nicht Prozeduren für alle möglichen Konstellationen beim Transfer von Textdaten aus einer Quelle in eine andere beschrieben werden. Wir können hier nur nachdrücklich darauf hinweisen, dass Sie bei allen Speicher- und Kopiervorgängen grundsätzlich Ihre Aufmerksamkeit auf diese potentielle Fehlerquelle richten sollten. Es ist überdies empfehlenswert, stets Sicherungskopien in mehreren Versionen (1 Tag, 3 Tage, 1 Woche) vorzuhalten (vgl. Anm. **Fehler! Textmarke nicht definiert.**).

3.2.4 Das Betacode-Verfahren

Das ↗Betacode-Verfahren basiert auf der Grundidee, beliebige Zeichen durch eine Abfolge von (lateinischen) Buchstaben und Zeichen zu kodieren, deren Zahlenwert im Windows-↗Betriebssystem – und nicht nur dort; in diesem Punkt besteht zwischen *allen* ↗Betriebssystemen Übereinstimmung – zwischen 0 und dez127 liegt (sog. ↗ASCII-Code; vgl. http://de.wikipedia.org/wiki/Ascii). Eine Option im Rahmen dieser Kodierung ist die Verwendung von „↗HTML-Entities“ (s. unten Beispiel 3; vgl. auch http://de.selfhtml.org/html/referenz/zeichen.htm).

Beispiele:

(1) *MH=NIN A)/EIDE QEA\ *PHLHI+A/DEW *)AXILH=OS (erster Vers der Ilias; Kodierung nach TLG-Beta)

(2) \aem \'1v\ae\ngt\swr \'1fl\efj\swn d\sw g\sw\'1dr\ef\c,t \'1bl\aed\swr \aen d\swr \'1laft \sw\'1r\aem. (Wenkersatz 1 in siebenbürgisch-sächsischem Dialekt; Kodierung nach den Konventionen des Programmes „Praat“; s. unten S. 114; für phonetische Transkriptionen bietet sich auch die SAMPA-Kodierung an [vgl. http://de.wikipedia.org/wiki/SAMPA])

(3) Es grünt so grün, wenn Spaniens Blüten blüh'n. (Kodierung mit sog. „↗HTML-Entities“; dies ist das Verfahren, das ursprünglich für die Verwendung im Internet entwickelt wurde)

Das Beta-Code-Verfahren wirkt auf den ersten Blick nicht besonders attraktiv, ist aber ausgesprochen robust, da die Zahlen zwischen 0 und 127 auf allen gängigen ↗Betriebssystemen jeweils identischen Buchstaben bzw. Zeichen zugeordnet sind, d.h. Daten können ohne jedes Prob-

lem zwischen unterschiedlichen ↗Betriebssystemen ausgetauscht werden, ohne dass die Kodierung durcheinander gerät. Ein **wesentlicher Vorteil** dieses Verfahrens (u.a. gegenüber der Verwendung von ↗Unicode/↗UTF-8) ist, dass sämtliche benötigten Zeichen bequem über jede Standardtastatur eingegeben werden können. Ein gewisser **Nachteil** besteht darin, dass man bei der Suche nach Textmustern die ungewohnte Kodierung berücksichtigen muss, was besonders bei der Verwendung von Regulären Ausdrücken recht umständlich sein kann. Probleme bzw. ungewollte Ergebnisse können auch bei Sortierungen auftreten.

Wir empfehlen, den Datenbestand während der Bearbeitungsphase in dieser Kodierung zu belassen. Zum Zweck der Präsentation bzw. Publikation oder auch Recherche lassen sich die Daten durch Verwendung geeigneter Ersetzungsroutinen in nahezu beliebige Kodierungsformate überführen (z.B. ↗UTF-8).

3.2.5 Unicode und UTF-8

↗Unicode ist zunächst nichts anderes als eine weithin verbindliche ↗Tabelle von Zahlen, die jeweils exakt einem Zeichen zugeordnet sind:

```
65   LATIN CAPITAL LETTER A
66   LATIN CAPITAL LETTER B
67   LATIN CAPITAL LETTER C
...
913  GREEK CAPITAL LETTER ALPHA
914  GREEK CAPITAL LETTER BETA
915  GREEK CAPITAL LETTER GAMMA
...
3372 MALAYALAM LETTER BA
3373 MALAYALAM LETTER BHA
3374 MALAYALAM LETTER MA
```

Ein ↗Unicode-fähiges ↗Betriebssystem/Programm stellt jeweils das der Zahl zugeordnete Zeichen dar. Voraussetzung ist jedoch, dass das jeweilige ↗Betriebssystem über einen ↗Zeichensatz verfügt, der die passende ↗Glyphe zur Verfügung stellt. Ist das nicht der Fall, werden an der entsprechenden Stelle im Text gerne kleine Quadrate oder auch Fragezeichen dargestellt. Dieses Phänomen hat keinerlei substantielle Relevanz,

sondern bewegt sich ausschließlich auf der Ebene der graphischen Oberfläche.[15]

Computer arbeiten intern ausschließlich mit dem binären Zahlensystem, d.h. alle Zahlen werden durch Nullen (0) und Einsen (1) dargestellt. Aus historischen Gründen hat sich flächendeckend durchgesetzt, dass jeweils acht Positionen (bestehend aus Nullen und Einsen) *eine* Zahl repräsentieren:

> Beispiel: Im Computer wird die dezimale Zahl 86 folgendermaßen dargestellt: `01010110`

Bei der Beschränkung auf insgesamt acht Positionen lassen sich auf diese Weise lediglich die dezimalen Zahlen von 0 bis 255 (insgesamt also 256) darstellen. Das Problem besteht nun darin, dass die ↗Unicode-↗Tabelle für die Kodierung weit überwiegend Zahlen verwendet, die jenseits von 255 liegen.[16] Die Darstellung beispielsweise des oben erwähnten malayischen Zeichens „Ma“ mit der dezimalen Zahl 3374 ist bei der Beschränkung auf acht Positionen (Nullen und Einsen) demnach nicht möglich. In binärer Schreibweise sähe diese Zahl nämlich folgendermaßen aus: `1101 00101110` – würde also exakt zwölf Positionen mit Nullen und Einsen erfordern.

Die Lösung dieses Problems kann nur darin bestehen, jeweils mehr als ein ↗Byte pro darzustellendem Zeichen zu verwenden. Bei zwei ↗Byte ergeben sich rein rechnerisch schon 65536 kodierbare Zeichen und bei vier ↗Byte sogar 4.294.836.225, also über 4,2 Milliarden.

Wenn man also pro Zeichen jeweils zwei oder gar vier ↗Byte verwendet, resultiert bei der Kodierung von ausschließlich lateinischen bzw. englischen Buchstaben eine enorme Platz- bzw. Speicherverschwendung, da die betreffenden Zeichen alle mit dezimalen Zahlen zwischen 0 und dez127 kodiert sind und somit mit einem einzigen Block à acht Nullen und/oder Einsen darstellbar wären. Bei Verwendung von jeweils vier ↗Byte pro Zeichen, sähe das Wort „Baum“ im Speicher eines Computers in etwa folgendermaßen aus:

---

[15] Ein hervorragendes Hilfsmittel für die Verwendung von ↗Unicode ist die Internetseite http://www.fileformat.info.

[16] Die ↗Unicode-Version 6.2, veröffentlicht im September 2012, umfaßt insgesamt 110182 kodierte Zeichen aus 100 Schriftsystemen (https://de.wikipedia.org/wiki/Unicode; http://www.unicode.org/versions/Unicode6.2.0/)

```
00000000 00000000 00000000 01000010   B
00000000 00000000 00000000 01100001   a
00000000 00000000 00000000 01110101   u
00000000 00000000 00000000 01101101   m
```

Wie man sieht, bestehen die jeweils drei linken Blöcke nur aus Nullen, sind also gleichsam „leer“. Das Wort „Baum“ könnte also ohne weiteres auch so kodiert werden (jeder Buchstabe wieder von geschweiften Klammern umrahmt):

```
01000010     B
01100001     a
01110101     u
01101101     m
```

Da aber, wie schon gesagt, andere Zeichen, die durch dezimale Zahlen jenseits der 255 kodiert sind, mit jeweils einem Block nicht darstellbar sind, ist man auf die Idee verfallen, mit einer variablen Anzahl an Blöcken pro zu kodierendem Zeichen zu arbeiten. Dieses Verfahren wird als ↗UTF-8 bezeichnet. Die Abfolge eines lateinischen „a“ gefolgt von einem griechischen Gamma und dem malayischen Buchstaben Ma sähe dann folgendermaßen aus:

```
01100001  a
11001110 10110011  γ
11100000 10110100 10101110  ⍢
```

Die Berechnung der Blöcke des Gamma und des Ma folgt einer eigenen Systematik, die in einem Schaubild im Anhang erläutert ist (S. 202 Abschnitt 1.1). Angemerkt sei, dass Blöcke zusammengenommen jeweils *nicht* die dem Gamma bzw. dem Ma zugeordneten Zahlen ergeben (nur ein Teil davon). Wesentlich ist, dass das Auftreten von Nullen und Einsen am Anfang des linken Blocks die Anzahl der nachfolgenden Blöcke bezeichnet, die den nächsten Buchstaben repräsentieren. Beginnt ein Block mit einer 0, wissen geeignete Systeme, dass das Zeichen nur durch diesen einen Block repräsentiert wird. Im Fall des Gamma zeigen die beiden Einsen am Beginn des ersten Blocks an, dass das Zeichen durch insgesamt zwei Blöcke repräsentiert wird.

3.2.5.1 Probleme und Gefahren

Die Verwendung der ↗Unicode-Kodierung in Verbindung mit dem ↗UTF-8-Verfahren ist einesteils zwar sehr praktisch, weil sich damit ei-

ne enorme Menge an Schriftzeichen darstellen läßt, auf der anderen Seite sind damit jedoch auch Probleme und Risiken verbunden.

Da wäre zunächst das Problem der Zeicheneingabe über die Tastatur. Nur die wenigsten der nach ↗Unicode kodifizierten Zeichen lassen sich über eine westeuropäische Standard-Tastatur eingeben. Für die meisten Zeichen werden teils umständliche Modifizierungen der Tastaturbelegung bzw. speziell zu installierende Tastaturbelegungen benötigt.

Es sei ferner darauf hingewiesen, dass ↗Unicode keineswegs *alle* vorstellbaren bzw. erforderlichen Zeichen umfaßt. Es kann durchaus vorkommen, dass man Zeichen benötigt, die bislang noch nicht in der ↗Unicode-↗Tabelle berücksichtigt sind.

Eine wirkliche Gefahr bei der Verwendung von ↗Unicode bzw., genauer, ↗UTF-8 besteht darin, dass Texte durch unsachgemäßen Umgang regelrecht „zerstört“ werden können. Dazu folgendes Beispiel:

Bereits deutsche Umlaute werden in ↗UTF-8 mit jeweils zwei Blöcken à acht Nullen bzw. Einsen kodiert. Das Wort „für“ sieht in ↗UTF-8 so aus:

```
hexadezimal:  {66}      {c3  bc}  {72}
dezimal:  {102}     {195 188} {114}
```

Das „ü“ ist dargestellt durch die beiden Zahlen 195 und 188. Wenn man nun eine Datei mit diesem Inhalt von einem Editor öffnen läßt, der fälschlicherweise - entweder, weil nicht entsprechend konfiguriert,[17] oder, weil er nicht dazu in der Lage ist - davon ausgeht, dass die zu öffnende Datei in einer Kodierung vorliegt, in der grundsätzlich jeweils acht Nullen und Einsen genau ein Zeichen repräsentieren, dann entsteht folgendes Ergebnis:

```
fÃ¼r
```

Wenn man nun diese Zeichenfolge wiederum als ↗UTF-8 abspeichert, entsteht Folgendes:

```
hexadezimal:    {66}    {c3     83}    {c2     bc}  {72}
dezimal:        {102}   {195    131}   {194    188} {114}
```

[17] Der von uns empfohlene Editor VIM (s. S. 35) erhält bei Verwendung der im Anhang (s. S. 210) gelisteten Parameter (abzuspeichern in der Konfigurationsdatei _vimrc) die für die korrekte Darstellung von ↗UTF-8-kodierten Dateien erforderlichen Einstellungen.

Wie man sieht, sind bei dem Vorgang aus dem ursprünglichen „ü“ zwei neue Zeichen entstanden, die ihrerseits nun wieder als eigene Zeichen betrachtet und nach dem ↗UTF-8-Verfahren durch jeweils zwei „↗Bytes“ (= ein Block von acht Nullen und Einsen) kodiert werden. Diese Prozedur ist natürlich beliebig oft wiederholbar, so dass aus einem einzigen Zeichen schon nach wenigen Wiederholungen eine riesige Anzahl dann sinnloser Zeichen werden kann.

Besonders unerfreulich ist auch die Gefahr, dass man in einem größeren Text, dessen Verwandlung im eben dargestellten Sinn man zunächst nicht bemerkt hat, weiteren Text in der ↗UTF-8-Kodierung eingibt. Dies führt zu einer Inkonsistenz innerhalb der Datei. Um bei unserem Beispiel zu bleiben: Neben die fehlerhaft kodierten „fÃ¼r“ würden dann korrekt kodierte „für“ treten.

Es gilt also das bereits oben Gesagte. Beim Transfer von Texten und insbesondere beim Zusammenführen von Texten aus verschiedenen Quellen in eine Datei sind große Umsicht und regelmäßiges Kontrollieren der korrekten Kodierung zwingend erforderlich.

3.2.5.2 Kodierung des Zeilenendes

Eine Besonderheit stellt die Thematik der Kodierung des Zeilenendes in einer Datei dar. Zum Hintergrund muss man wissen, dass ein Zeilenende innerhalb einer Datei, die sich aus Sicht eines Computers als ununterbrochene Kette von Zahlen darstellt, genauso mit Zahlen kodiert werden muss wie jedes Textzeichen. Hinsichtlich des bzw. der Zahlenwerte, die jeweils für ein Zeilenende stehen, herrscht bis zum heutigen Tage zwischen den ↗Betriebssystemen Uneinheitlichkeit: Während Windows-Rechner grundsätzlich die Abfolge zweier Zahlen, nämlich x0d und x0a (dez13 und dez10), als Kodierung für das Zeilenende verwenden, schreiben Unix-↗Betriebssysteme (zu denen auch Mac OS X gehört) stets die Zahl x0a (dez10) allein.[18] Öffnet man ein Dokument, das auf einem Windows-Rechner abgespeichert wurde, mit einem Macintosh-

[18] Manche, heute kaum noch verwendete ↗Betriebssysteme wie z.B. ältere Apple-↗Betriebssysteme (bis Version 9), kodieren das Zeilenende auch mit einem einzelnen x0d. Auch wenn diese ↗Betriebssysteme selbst nicht mehr eingesetzt werden, kann es sein, dass es noch entsprechende Dateien gibt.

Rechner (unter OS X), dann erscheint an allen Zeilenenden eine Repräsentation des Zahlenwertes x0d (dez13), zumeist in Form eines „^M“.[19]

Umgekehrt scheint ein Text, der unter Mac OS X abgespeichert und dann mit einem Windows-Rechner geöffnet wurde, gar keine Zeilenumbrüche zu enthalten.

Um dies zu vermeiden, gibt es folgende Möglichkeiten: Bei Variante 1 (Windows ⇒ Mac) muss man entweder beim Abspeichern darauf achten, dass das spezielle MacOS/Unix-Format geschrieben wird, oder man entfernt aus der Windows-konformen Datei durch einen Ersetzungsbefehl alle „0d“ (z.B. sed ':a;N;$!ba;s/\n/ /g' [Dateiname]).

Bei Variante 2 (Mac ⇒ Windows) kann man wiederum beim Abspeichern auf die entsprechende Option achten, oder man ersetzt anschließend alle „0a“ durch „0d0a“.

Im Editor „VIM“ läßt sich die entsprechende Auswahl durch den Befehl „:se fileformat=unix“ bzw. „:se fileformat=dos“ treffen. In Microsoft Word erreicht man diese Option über den Befehl „Speichern unter ...“ ⇒ Option „nur Text“ (s. Abbildung 15).

Bei der Erstellung eines Textkorpus muss sorgfältig auf die Kodierung des Zeilenendes geachtet werden.[20] Der folgende Haiku wurde der Webseite https://de.wikipedia.org/wiki/Haiku entnommen und in jeweils einer Windows- und einer Unix-kodierten Textdatei abgespeichert. Das Unix-Kommando xxd erlaubt die hexadezimale Darstellung der Dateiinhalte (sog. „Hexdump“). Haiku:

```
Ab der Mittagszeit
ist es etwas schattiger
ein Wolkenhimmel
```

Windows: 0d 0a (= CR LF = \r\n) (Datei haiku_windows_0d0a):

[19] Für die Zahlenwerte 0d und 0a werden auch die Abkürzungen CR (= carriage return; "Wagenrücklauf", in Anlehnung an den Vorgang bei der Schreibmaschine) und LF (= line feed; "Zeilenvorschub", gleichfalls referierend auf die Schreibmaschine) verwendet. Die Schreibweisen ^M (0d, CR) und ^J (0a, LF) werden "Escape-Sequenzen" genannt.

[20] Das Beispiel ist dem unter der Adresse https://www.dh-lehre.gwi.uni-muenchen.de/?p=3900#subchapter:kodierung-des-zeilenendes abrufbaren Beitrag entnommen.

```
slu@PCROMLAB1301:Texte$xxd -g 1 haiku_windows_0d0a.txt
0000000: 41 62 20 64 65 72 20 4d 69 74 74 61 67 73 7a 65 Ab der Mittagsze
0000010: 69 74 0d 0a 69 73 74 20 65 73 20 65 74 77 61 73 it..ist es etwas
0000020: 20 73 63 68 61 74 74 69 67 65 72 0d 0a 65 69 6e schattiger..ein
0000030: 20 57 6f 6c 6b 65 6e 68 69 6d 6d 65 6c 0d 0a    Wolkenhimmel..
```

Unixsysteme inklusive Mac OS X: 0a (= LF = \n) (Datei haiku_unix_0a):

```
slu@PCROMLAB1301:Texte$xxd -g 1 haiku_unix_0a.txt
0000000: 41 62 20 64 65 72 20 4d 69 74 74 61 67 73 7a 65 Ab der Mittagsze
0000010: 69 74 0a 69 73 74 20 65 73 20 65 74 77 61 73 20 it.ist es etwas
0000020: 73 63 68 61 74 74 69 67 65 72 0a 65 69 6e 20 57 schattiger.ein W
0000030: 6f 6c 6b 65 6e 68 69 6d 6d 65 6c 0a             olkenhimmel.
```

In beiden Texten ist das Wort „schattiger“ unterstrichen. Im Fall der Windows-kodierten Datei besteht die Gefahr, dass der 0d-Wert bei einer automatisierten Tokenisierung durch ein Unix-System (oder auch eine Unix-Emulation wie Cygwin) als Bestandteil des Wortes „schattiger“ betrachtet wird. Eine entsprechende Tokenliste sähe dann folgendermaßen aus (die Zahlenwerte 20 [= Blank] und 0a fungieren als ↗Separatoren und sind daher verschwunden):

```
41 62                                           Ab
64 65 72                                        der
4d 69 74 74 61 67 73 7a 65 69 74 0d             Mittagszeit
69 73 74                                        ist
65 73                                           es
65 74 77 61 73                                  etwas
73 63 68 61 74 74 69 67 65 72 0d                schattiger
65 69 6e                                        ein
57 6f 6c 6b 65 6e 68 69 6d 6d 65 6c 0d          Wolkenhimmel
```

Eine Suche nach dem Wort „schattiger“ in o.a. Tokenliste würde keinen Treffer ergeben, da keine exakte Übereinstimmung in der numerischen Repräsentation zwischen Suchmuster und gespeichertem ↗Token vorhanden ist:

```
Suchmuster                      Token
73 63 68 61 74 74 69 67 65 72   73 63 68 61 74 74 69 67 65 72 0d
```

Dasselbe Phänomen/Problem träte im gegebenen Beispiel auch bei den ↗Tokens „Mittagszeit“ und „Wolkenhimmel“ auf.

3.3 Die „Shell“

Für die effiziente automatische Verarbeitung großer Datenmengen sind die gewohnten Windows-Programme mit ihrer graphischen Benutzeroberfläche nur sehr bedingt geeignet. In der Praxis hat sich die Verwendung von Programmen, die von der sogenannten „Kommandozeile“ aus gestartet und gesteuert werden, bestens bewährt. Bei der Kommandozeile handelt es sich um eine – auf den ersten Blick recht unattraktive – Möglichkeit, den Computer mit über die Tastatur einzutippenden Kommandos zu steuern. Jedes ↗Betriebssystem bietet diese Möglichkeit.

3.3.1 Windows-Shell

Auf Windows-PCs startet man die Kommandozeile z.B. (es gibt verschiedene Möglichkeiten) über die Tastenkombination „Windows + r“ („r“ für „run“) und das Eingeben des Befehls „cmd“:

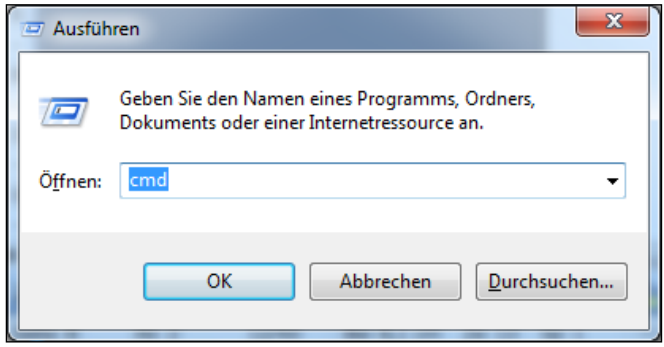


Abbildung 16: Start der Windows-↗Shell

Nach einem Klick auf „OK“ öffnet sich das Kommando-Fenster:

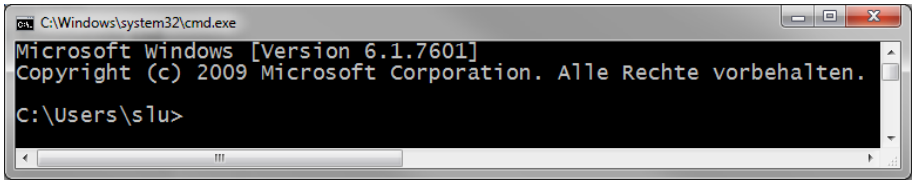


Abbildung 17: Die gestartete Windows-↗Shell

Ein blinkender ↗Cursor hinter dem sog. „Prompt“ (im Beispiel „C:\Users\slu“) zeigt an, dass der Computer auf die Eingabe von Befehlen wartet. Welche Befehle erlaubt bzw. möglich sind, hängt im Einzel-

nen von der Konfiguration des Computers ab, eine gewisse Anzahl von Kommandos ist jedoch den meisten Computern gemein. Eine erste, grobe Übersicht erhält man durch die Eingabe des Kommandos „help“ (getestet unter Windows 7). Hilfe zur Verwendungssyntax der einzelnen Kommandos erhält man durch die Eingabe von „ /?“ hinter dem Kommando. Jedes Kommando wird nach Drücken der Enter-Taste ausgeführt.

3.3.2 Cygwin und die Unix-Shell

Die oben beschriebene ↗Shell des Windows-↗Betriebssystems ist nach unserer Erfahrung in mancher Beziehung unpraktisch. Die besten Erfahrungen haben wir mit einer ↗Shell des Unix-↗Betriebssystems gemacht, die wir aus diesem Grund derzeit empfehlen möchten.

Auf genuinen Unix-Systemen, zu denen auch Apple-Computer ab Mac OS X gehören, steht eine Unix-↗Shell automatisch zur Verfügung,[21] und auch auf Windows-PCs läßt sich eine Unix-↗Shell verwenden: Das Programm „Cygwin“[22] emuliert auf einem Windows-PC ein Unix-System, d.h. das Programm tut so, als ob der Rechner als ↗Betriebssystem Unix installiert hätte, obwohl es sich um Windows handelt.[23] Der wesentliche Grund für die Verwendung einer ↗Shell ist, dass sich Vorgänge damit besonders leicht automatisieren lassen (mit ↗Shell-Skripts und job-Dateien).

Beispiele:

- Sie wollen Dateien nach einer bestimmten komplexen Systematik benennen
- Sie wollen bestimmte Zeichenketten (↗Strings) suchen und das Suchergebnis in einer eigenen Datei abspeichern
- Sie wollen Passagen aus Dateien extrahieren und in anderen Dateien zusammenfügen.
- Sie wollen statistische Operationen durchführen

[21] Auf Apple-Rechnern mit Mac OS X startet man die Shell mit dem Programm „Terminal“ im Ordner „Dienstprogramme“. Auf manchen Unix-Systemen heißt das ↗Shell-Programm auch „Konsole“.

[22] Downloadbar unter http://www.cygwin.com. Eine gute Kurzanleitung für die Installation von cygwin wurde vom Lehrstuhl Mathematik III der TU Dortmund verfasst. Das Dokument kann unter folgendem Link heruntergeladen werden: http://www.kit.gwi.uni-muenchen.de/wp-content/uploads/CygwinInstallation.pdf

[23] Die im folgenden beschriebenen Prozeduren sind alle unter Cygwin entwickelt bzw. getestet. Sie sind nicht unbedingt auf genuine Unix-Systeme übertragbar, sondern bedürfen dafür unter Umständen kleinerer Anpassungen.

- Sie wollen Wortlisten erstellen und analysieren

Um diese Aufgaben möglichst effizient erledigen zu können, empfiehlt es sich grundsätzlich, das Datenmaterial in möglichst flachen Hierarchien zu verwalten. Vermeiden Sie die Verteilung der Dateien in ein verschachteltes System von Ordnern. Nutzen Sie stattdessen die Möglichkeit, Ordnungskriterien in kodierter Form in die Namen der Dateien zu integrieren.

ACHTUNG: Die Cygwin-/Unix-↗Shell fragt in den seltensten Fällen nach, ob Sie eine bestimmte Operation *wirklich* ausführen wollen! Gehen Sie daher mit dem System bedächtig um und halten Sie stets Sicherungskopien bereit![24]

3.3.2.1 Grundlegende Operationen

Die Cygwin/Unix-↗Shell wird durch Doppelklick auf das entsprechende Dateisymbol gestartet. Anschließend erscheint ein Fenster, in dem (reichlich unspektakulär) ein „↗Cursor“ blinkt. Über dem ↗Cursor erscheint die Information, mit welcher Kennung Sie an welchem Rechner angemeldet sind, sowie der Name des aktuellen Verzeichnisses, in dem Sie sich befinden.

Den Namen des jeweils aktuellen Verzeichnisses können Sie sich auch mit dem Kommando „pwd“ (= „print working directory“) ausgeben lassen. Eine Übersicht, welche Dateien sich in diesem Verzeichnis befinden, erhalten Sie mit dem Kommando „ls“ (= „list“).

Fast alle Unix-Kommandos[25] können in ihrem Verhalten durch die Verwendung von Optionen beeinflußt werden. Diese Optionen werden in aller Regel unmittelbar hinter dem Kommando mit einem vorangestellten Minuszeichen angegeben. Auch die Kombination mehrerer Opti-

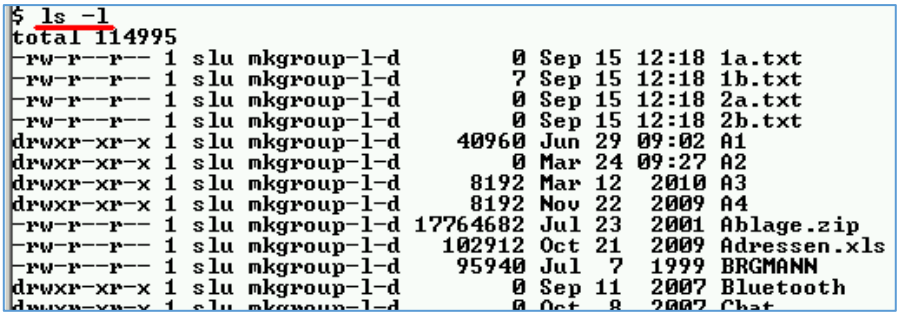
[24] Für die richtige Art und Weise, Sicherungskopien anzufertigen, gibt es verschiedene Strategien. Welche die richtige ist, hängt von der Art der zu sichernden Daten und den zur Verfügung stehenden Ressourcen ab. Grundsätzlich gilt, dass Sicherungskopien auf externen, d.h. nicht im Arbeitscomputer verbauten, Medien wie z.B. externen Festplatten, USB-Sticks oder Serverlaufwerken abgelegt werden. Für hochrelevante und permanenter Änderung unterworfene Daten erscheint uns die Sicherungsstrategie „Türme von Hanoi“ besonders empfehlenswert. Dabei werden die Daten reihum in unterschiedlichen Zeitintervallen auf mindestens drei verschiedene Speichermedien geschrieben. Eine gute Beschreibung des Prinzips findet sich z.B. unter https://de.wikipedia.org/wiki/Datensicherung#T.C3.BCrme_von_Hanoi.

[25] Übersichten über in der ↗Shell verfügbare Kommandos erhält man z.B. durch Eingabe der Befehle „help“ und „info“. Auch im Internet finden sich zahlreiche Zusammenstellungen und Erläuterungen von ↗Shell-Kommandos.

onen ist möglich. Eine Übersicht über die jeweils möglichen Optionen erhalten Sie durch Aufruf der Hilfefunktion, die praktisch jedem Kommando beigegeben ist. Diese kann entweder durch „[kommando] --help“ oder durch „man [kommando]“ aufgerufen werden.

Ruft man z.B. das Kommando ls mit der Option -l auf, erhält man ausführlichere Informationen (im wesentlichen Dateigröße und Datum der letzten Änderung) zu den im Verzeichnis enthaltenen Dateien:

Beispiel:



Um sich im ↗Dateisystem zu bewegen, sprich: um in andere Verzeichnisse zu wechseln, gibt es das Kommando „cd“ (change directory). Vom aktuellen Verzeichnis kann man sozusagen „nach oben“ oder auch „nach unten“ wandern. Der Weg „nach oben“ kann schrittweise in das jeweils unmittelbar oberhalb gelegene Verzeichnis erfolgen. Für dieses Verzeichnis gibt es die Abkürzung „..“. Die Abkürzung für das aktuelle Verzeichnis lautet „.“.

Beispiel: cd ..

Der Wechsel in tiefergelegene Verzeichnisse erfolgt durch die Angabe des jeweiligen Namens:

```
cd [Verzeichnisname]
```

Das oberste Verzeichnis ist ansprechbar mit „/“.

```
cd /
```

In der Cygwin-Installation ist dieses oberste Verzeichnis grundsätzlich identisch mit dem Verzeichnis, in dem Cygwin auf dem Windows-Rechner installiert ist (zumeist: C:\Programme\cygwin). Sämtliche auf

dem Windowsrechner vorhandenen Laufwerke sind dann über das Verzeichnis „cygdrive“, gefolgt von dem jeweiligen Windows-Laufwerksbuchstaben erreichbar.

cd /cygdrive/c wechselt auf die oberste Ebene des Windows-Laufwerks c:

Angenommen, die Dateien, mit denen man arbeiten möchte, befinden sich im Verzeichnis d:\corpus\italien, dann wechselt man auf folgende Weise in dieses Verzeichnis:

```
cd /cygdrive/d/corpus/italien
```

Erst wenn dieser Befehl gegeben wurde, kann man mit dem in diesem Verzeichnis befindlichen Material arbeiten, ohne bei Kommandos jeweils den vollständigen Pfad mit angeben zu müssen.

Um eine Übersicht der im Verzeichnis enthaltenen Dateien zu erhalten, gibt man wiederum den Befehl „`ls -l`“ ein.

Zur Erstellung eines neuen Verzeichnisses gibt es den Befehl „mkdir“ („make directory“): `mkdir [Verzeichnisname]`.

Der Befehl „rmdir“ („remove directory“) entfernt ein Verzeichnis (das Verzeichnis muss leer sein!).

Der Befehl rm löscht Dateien.

Der Befehl cp kopiert Dateien.

Beispiel:

```
cp datei.txt unterverzeichnis/
```

cp datei.txt unterverzeichnis/neuerName.txt benennt die Datei beim Kopiervorgang gleichzeitig um.

Der Befehl mv („move“) funktioniert genauso wie cp, nur dass die ursprüngliche Datei gelöscht bzw. der Dateiname geändert wird.

Um mehr als eine Datei ansprechen zu können, gibt es Platzhalter („wildcards“): Das Fragezeichen („?“) steht für genau ein beliebiges Zeichen im Namen einer Datei (oder eines Verzeichnisses), der Stern („*“) für eine beliebige Anzahl von Zeichen.

Beispiel:

Sie wollen nur Text-Dateien anzeigen, die (wir beziehen uns auf das oben, S. 25 vorgestellte Beispiel des AsiCa-Korpus) weiblichen Informanten zugeordnet sind:

```
ls ??w??.txt
```

`cp 60?2?.* auswahl/` würde sämtliche Dateien (Bild-, Text-, Audio-), die den 60er-Jahren und dem Bildungsgrad 2 zuzuordnen sind, in das Verzeichnis „auswahl“ kopieren.

Alle Dateien eines Verzeichnisses, die Dateiname und Dateiextension durch „.“ trennen, werden mit *.* angesprochen. Alle Dateien eines Verzeichnisses werden mit * angesprochen.

3.3.2.2 Ein- und Ausgabekanäle und deren Umlenkung

Eine ↗Shell kennt drei Kanäle für den Datenfluss: Den Standardeingabe- (stdin), Standardausgabe- (stdout) und Standardfehlerkanal (stderr).

Normalerweise erfolgt die Eingabe von Befehlen und/oder Text über die Tastatur (Standardeingabe). Alles, was Cygwin produziert, wird auf dem Standardausgabekanal, dem Bildschirm (im Cygwin-Fenster) ausgegeben und ist unter Umständen schon dann „verloren“, wenn der Text durch die Ein-/Ausgabe weiterer Befehle/Ergebnisse am oberen Rand des Cygwin-Fensters verschwunden ist. Der Standardfehlerkanal ist nur im Rahmen der ↗Shell-Programmierung wichtig; auf ihn soll hier nicht näher eingegangen werden.

Alle Kanäle lassen sich mit Hilfe eines speziellen Mechanismus der ↗Shell umlenken. Dazu stehen die Operatoren „<“, „>“ und „>>“ zur Verfügung. Sämtlicher Text, der von Cygwin normalerweise auf dem Bildschirm ausgegeben wird, lässt sich z.B. in eine Datei umleiten. Diese Umleitung kann auf zwei Weisen erfolgen: Zum einen ist es möglich, den Text in eine Datei schreiben zu lassen. Dies erreicht man mit dem Operator „>„, der die Zieldatei neu anlegt und zugleich befüllt.

`ls > verzeichnisinhalt.txt` schreibt den Text, den ls auf dem Bildschirm ausgeben würde, in die Datei verzeichnisinhalt.txt.

`ls >> verzeichnisinhalt.txt` dagegen würde den Text bei jedem weiteren Aufruf an eine schon bestehende Datei verzeichnisinhalt.txt anhängen.

Die andere Möglichkeit ist die Verwendung der sog. „↗Pipe“ (dahinter steckt die Metapher der Pipeline, durch die Material von einem Ende zum anderen transportiert wird). Der entsprechende Operator ist der senkrechte Strich (auf deutschen Tastaturen für gewöhnlich durch die Tastenkombination AltGr + Größer-/Kleinertaste erreichbar), der auch selbst als „↗Pipe“ bezeichnet wird. Mit der ↗Pipe wird die Ausgabe eines Kommandos unmittelbar zu Eingabe eines weiteren Kommandos, wobei auch längere Ketten gebildet werden können. Beispiel:

```
ls | less
```

ls erzeugt das Inhaltsverzeichnis des aktuellen Verzeichnisses und übergibt den entsprechenden Text direkt an das Kommando „less“. less ist ein einfacher „Viewer“, also ein Programm, der nichts anderes tut, als Text darzustellen.

Auch die Kombination von ↗Pipe und Schreibumlenkung ist möglich:

```
ls | grep 'muster'  > ergebnis.txt
```

In diesem Beispiel wird das Inhaltsverzeichnis an das Kommando grep übergeben, das eine Mustersuche bewerkstelligt. Hier werden ↗Zeilen herausgefiltert, die die Zeichenfolge „muster“ enthalten. Das herausgefilterte Ergebnis wird anschließend in die Datei ergebnis.txt geschrieben.

So wie das Schreiben in Dateien ist auch das Lesen aus Dateien möglich. Der entsprechende Operator ist das Kleiner-Zeichen („<“). Beispiel:

```
tr a A < beispiel.txt > ergebnis.txt
```

Das Kommando tr ersetzt ein gegebenes Zeichen durch ein anderes (s.u.). Der Text, auf den dieses Kommando angewendet werden soll, wird aus der Datei beispiel.txt gelesen. tr ersetzt alle in diesem Text vorkommenden „a“ durch „A“. Durch den Umlenkungsmechanismus der Shell wird das Ergebnis in die Datei ergebnis.txt geschrieben.

3.3.2.3 Nützliche Befehle

Unter den sog. UNIX-Tools werden eine Fülle von Programmen zusammengefasst, aus denen im Folgenden die aus Sicht eines Korpuslinguisten wichtigsten in alphabetischer Reihung zusammengestellt sind.

- ascii Durch Eingabe des Kommandos ascii wird im ↗Terminal eine ASCII-↗Tabelle ausgegeben.

- awk s. (g)awk

- cat „concatenate“ („konkatenieren“) Dateiinhalt ausgeben, Dateien verbinden. Beispiel: cat string.txt quodlibet.txt verkettet den Inhalt der beiden angebenen Dateien, indem er ihn unmittelbar hintereinander schreibt, und gibt ihn auf dem Bildschirm aus. Es können beliebig viele Dateien auf diese Weise miteinander verbunden werden. Durch Umleitung in eine neue Datei erreicht man die Vereinigung der beiden Ausgangsdateien:

```
cat string.txt quodlibet.txt > neu.txt
```

- cut cut schneidet von jeder ↗Zeile eine zu definierende Anzahl von Buchstaben/Zeichen oder Spalten aus. Beispiel: `cut -c 4-10 text.txt` schneidet aus jeder ↗Zeile in der Datei text.txt die Zeichen 4 bis 10 aus und gibt sie auf dem Bildschirm aus. ACHTUNG: cut ist nicht „multi-byte-fähig“, d.h. es erkennt nicht, wenn – wie dies z.B. bei ↗UTF-8-kodierten Dateien der Fall ist – ein Buchstabe aus mehr als einem ↗Byte besteht (cut betrachtet also das eine Zeichen als deren zwei, was zu unsinnigen Ergebnissen führt)! Sinn macht daher die Verwendung von cut *nur*, wenn die Ausgangsdatei als ↗Tabelle organisiert ist. Mit der Option -f weist man cut an, nach den ↗Feldtrennern (Element, das die Spalten einer ↗Tabelle voneinander abgrenzt, meist Spatium oder Tabulator) in jeder ↗Zeile zu suchen. Standardmäßig betrachtet cut den Tabstop als ↗Feldtrenner. Mit der Option -d („delimiter“) können aber auch beliebige andere Zeichen als ↗Feldtrenner definiert werden (etwa „;“ oder „,“, wie bei Dateien im csv-Format üblich).

  Beispiel:

```
cut -f1,3 tabelle.txt > d1.txt
```

schneidet aus der Datei tabelle.txt die Felder (Spalten) 1 und 3 jeder ↗Zeile aus und schreibt sie in die Datei d1.txt.

- echo Gibt eine Zeichenkette („↗String“) auf dem Bildschirm aus.

Beispiel:

```
echo "Hallo Welt!"
```

schreibt auf dem Bildschirm "Hallo Welt!".

```
echo „Hallo Welt!“ > string.txt
```

schreibt „Hallo Welt!“ in die Datei string.txt.

- exit Das Kommando exit schließt das Fenster der ↗Shell und beendet den entsprechenden Prozess. Speziell bei Verwendung der Cygwin-↗Shell ist *unbedingt* darauf zu achten, die ↗Shell auf diese Weise zu beenden, da „exit“ nicht nur das Schließen der ↗Shell bewirkt, sondern gleichzeitig dafür sorgt, dass die Liste der bis dahin auf der Kommandozeile eingegebenen Befehle in die im ↗Homeverzeichnis gelegene Datei .bash_history geschrieben wird, so dass diese bei Aufruf eines neuen ↗Shell-Fensters bzw. -Prozesses wieder zur Verfügung stehen. Schließt man das Cygwin-Fenster in der für Windows üblichen Weise durch einen Klick auf das kleine Kreuz am rechten oberen Fensterrand, gehen die im Lauf der Sitzung eingegebenen Befehle unwiederbringlich verloren.

- file „file“ liefert Informationen zur Art einer Datei und, im Fall, dass es sich um eine Textdatei handelt, zur darin verwendeten Zeichen- und Zeilenende-Kodierung.

Beispiel:

```
file *
```

bewirkt (unter der Annahme, dass sich die folgenden Dateien und Unterverzeichnisse im aktuellen Verzeichnis befinden) folgende Ausgabe. Die linke Spalte zeigt den Objektnamen, die rechte gibt das Datenformat des Objekts an:

```
_viminfo:        ISO-8859 text
_vimrc:          UTF-8 Unicode text
Ablage:          directory
Admin:           directory
Archiv:          directory
befehle.sql:     UTF-8 Unicode text, with very long lines,
                 with CRLF line terminators
```

```
Beispiel.csv:    UTF-8 Unicode text, with CRLF line
                 terminators
Beispiel.xlsx:   Microsoft Excel 2007+
Beispiele:       directory
config.zip:      Zip archive data, at least v2.0 to extract
Cygwin.bat:      DOS batch file, ASCII text, with CRLF line
                 terminators
Download:        directory
heute.txt:       ISO-8859 text
hui.txt:         UTF-8 Unicode text
Literature.xlsx: Microsoft Excel 2007+
abc.sql:         UTF-8 Unicode text, with very long lines
kml_convert.sh:  POSIX shell script, ASCII text executable
```

- (g)awk awk ist ein Kommandointerpreter, der die Befehle der gleichnamigen Programmiersprache ausführen kann. Die Programmiersprache AWK ist sehr mächtig und wird weiter unten (S. 87ff.) ausführlicher dargestellt. Das Programm awk kann zur Bewältigung kleinerer Aufgaben direkt auf der Kommandozeile verwendet werden. Die auszuführenden Befehle werden dabei hinter das Kommando geschrieben, eingeschlossen in Hochkommata (') und geschweifte Klammern ({}). Den zu bearbeitenden „input“ übernimmt awk entweder von einer ↗Pipe oder liest ihn aus einer Datei.

awk existiert in unterschiedlichen Implementierungen (awk, nawk, gawk, von denen alle wieder in unterschiedlichen Versionen vorliegen), die hinsichtlich des zur Verfügung stehenden Befehlsrepertoires leicht von einander abweichen. So kennt das Programm gawk z.B. den Befehl „asort“ (zur Sortierung von Zeichen[ketten]), awk hingegen nicht.

Beispiel:

Die Textdatei eichendorff.txt enthält einen ↗Fließtext. Der folgende Aufruf verwandelt diesen in eine ↗Tabelle:

```
less eichendorff.txt | awk '{for (i=1;i<=NF;i++) {print
"Zeile " NR, "Wort Nr. " i, $i}}' > tabelle.txt
```

Der Inhalt der dabei erzeugten Datei tabelle.txt sieht dann wie folgt aus:

```
Zeile 1 Wort Nr. 1 Das
Zeile 1 Wort Nr. 2 Rad
Zeile 1 Wort Nr. 3 an
Zeile 1 Wort Nr. 4 meines
Zeile 1 Wort Nr. 5 Vaters
Zeile 1 Wort Nr. 6 Mühle
```

```
Zeile 1 Wort Nr. 7 brauste
Zeile 1 Wort Nr. 8 und
Zeile 1 Wort Nr. 9 rauschte
Zeile 1 Wort Nr. 10 schon
Zeile 1 Wort Nr. 11 wieder
Zeile 1 Wort Nr. 12 recht
Zeile 1 Wort Nr. 13 lustig,
Zeile 1 Wort Nr. 14 der
Zeile 1 Wort Nr. 15 Schnee
Zeile 2 Wort Nr. 1 tröpfelte
Zeile 2 Wort Nr. 2 emsig
Zeile 2 Wort Nr. 3 vom
Zeile 2 Wort Nr. 4 Dache,
Zeile 2 Wort Nr. 5 die
Zeile 2 Wort Nr. 6 Sperlinge
Zeile 2 Wort Nr. 7 zwitscherten
Zeile 2 Wort Nr. 8 und
Zeile 2 Wort Nr. 9 tummelten
Zeile 2 Wort Nr. 10 sich
Zeile 2 Wort Nr. 11 dazwischen;
```

- grep „global/regular expression/print" (https://de.wikipedia.org/wiki/Grep). Zeichenketten bzw. reguläre Ausdrücke (s. unten S. 81) in Dateien suchen und ausgeben lassen.

Beispiel: Das Kommando

```
grep -n Eichendorff w1.txt > ergebnis.txt
```

sucht die Zeichenfolge „Eichendorff" in der Datei w1.txt und schreibt die ↗Zeilen, in denen die Zeichenfolge auftritt, in die Datei ergebnis.txt. Die Option -n bewirkt, dass vor jede ↗Zeile deren Nummer, bezogen auf die durchsuchte Datei, geschrieben wird. Die Verwendung von Platzhaltern (*,?) ist möglich, auch das Durchsuchen von Unterverzeichnissen eines gegebenen Verzeichnisses:

```
grep -R 'Hallo Welt!' *.txt
```

sucht den ↗String „Hallo Welt!" in allen Dateien mit der Extension „txt" im aktuellen sowie allen Unterverzeichnissen (Option „-R": rekursiv). Zeichenfolgen, die ein oder mehrere Spatien enthalten (wie im gegebenen Beispiel), müssen von Hochkommata umschlossen sein.

- head Das Kommando head gibt standardmäßig die ersten zehn ↗Zeilen einer Datei aus. Als Option kann die Anzahl der auszugebenden ↗Zeilen explizit angegeben werden:

```
head -25 textdatei.txt
```

- history Der Befehl history ruft die Liste der zuletzt auf der Kommandozeile eingegebenen Befehle auf. Die Liste ist numeriert. Einzelne Befehle der Liste können erneut aufgerufen werden, indem die dem Befehl vorangestellte Zahl zusammen mit einem vorangestellten Ausrufezeichen als Befehl auf der Kommandozeile eingetippt werden. Besonders sinnvoll ist die Verwendung des history-Befehls in Verbindung mit dem grep-Befehl, weil man auf diese Weise leicht komplexe Befehle wiederfinden und erneut aufrufen kann, die man in der Vergangenheit schon einmal verwendet hat. Beispiel:

```
history | grep wget
```

Als Ergebnis erhält man all die Kommandoaufrufe, in denen der Befehl wget vorkommt:

```
463  wget
466  wget -q -Q10m -r -l3 -N -O - -R avi,mpg,mp3,exe -A
      htm,html,txt http://www.repubblica.it
467  wget -q -Q10m -r -l3 -R -N -O - avi,mpg,mp3 -A
      htm,html,txt http://www.repubblica.it | grep
      'Obama'
493  wget -m -p -k http://asica.itg.uni-muenchen.de
494  wget -m -p -k http://asica.gwi.uni-muenchen.de
636  history | grep wget
637  wget -m -p -k http://www.comuni-italiani.it
639  wget -m -p -k http://www.comuni-italiani.it
700  history | grep wget
838  history | grep wget
```

Das Kommando !493 würde dann den Befehl „`wget -m -p -k http://asica.itg.uni-muenchen.de`“ ausführen. VORSICHT: Die Ausführung des Kommandos erfolgt ohne Rückfrage!

Die Tastenkombination ctrl+r ermöglicht die direkte Suche im history-File. Schreibt man nach Drücken von ctrl+r Teile von Befehlsfolgen, werden, beginnend mit den jüngsten, passende Einträge in der History-Datei auf der Kommandozeile angezeigt. Erneutes Drücken von ctrl+r springt zum nächst älteren Treffer in der History-Datei. VORSICHT: Drü-

cken der Enter-Taste führt den gerade angezeigten Befehl ohne Rückfrage aus! ctrl+j kopiert den aktuellen Treffer auf die Kommandozeile, so dass er zunächst modifiziert werden kann.

- less Gibt den Inhalt einer Datei auf dem Bildschirm aus.

Beispiel:

```
less string.txt
```

öffnet die Datei string.txt und zeigt (s. Beispiel oben) „Hallo Welt!“ an. less gestattet es, sich in der Datei zu bewegen (Sprung ans Ende oder an den Anfang). Auch Suche nach regulären Ausdrücken ist möglich (Taste „/“). Das Drücken der Taste „h“ zeigt sämtliche verfügbaren Optionen an. Zum Verlassen von less drückt man die Taste „q“ („quit“).

- man Fast alle in der Unix-/Cygwin-↗Shell verwendbaren Befehle besitzen sog. „Manual-Pages“, in denen die jeweilige Kommando-Syntax dokumentiert ist und häufig auch Syntaxbeispiele zu finden sind. Diese Manual-Pages können durch Eingabe des Befehls „man“ gefolgt vom Namen des gewünschten Kommandos. Beispiel:

```
man grep
```

- nl Das Kommando nl schreibt vor jede ↗Zeile der als Parameter genannten Datei(en) deren Nummer. Die Ausgabe erfolgt auf dem Bildschirm und kann, wie üblich, in eine Datei umgelenkt werden. Die Zeilennummern sind dann integrativer Bestandteil des Textes!

Beispiel:

Vorher:

```
-- sql-Abfrage
-- Autor: slu
-- Datum: 15.3.12
-- Zweck: Ueberblick ueber Erfassung Sardinien
```

Kommando:

```
nl datei.txt
```

Nachher:

```
1  -- sql-Abfrage
2  -- Autor: slu
3  -- Datum: 15.3.12
4  -- Zweck: Ueberblick ueber Erfassung Sardinien
```

- od od steht für „octal dump“, also „oktale Ausgabe“. od, gefolgt vom Namen einer Datei, gibt den Inhalt der Datei in Zahlengestalt aus, wobei in der Standardeinstellung für die Darstellung der Zahlen das oktale Zahlensystem verwendet wird. Da alle Dateien in ihrer wahren Gestalt aus (eigentlich binären) Zahlen bestehen, erlaubt der Befehl od den Blick auf eben jene wahre Gestalt, was besonders dann von Nutzen ist, wenn man Problemen z.B. im Zusammenhang mit der Suche nach Zeichenmustern auf den Grund gehen möchte.

  Beispiel:

  Die Text-Datei eichendorff.txt enthält folgenden Text:

```
Das Rad an meines Vaters Mühle brauste und rauschte scho
n wieder recht lustig, der Schnee
tröpfelte emsig vom Dache,
die Sperlinge zwitscherten und tummelten
sich dazwischen;
```

Das Kommando „`od -A x -t x1 eichendorff.txt`“ mit der Option „`-t x1`“ bewirkt, dass die Zahlen nicht im oktalen, sondern im hexadezimalen Zahlensystem ausgegeben werden, und liefert folgendes Ergebnis:

```
0000000 44 61 73 20 52 61 64 20 61 6e 20 6d 65 69 6e 65
0000010 73 20 56 61 74 65 72 73 20 4d c3 bc 68 6c 65 20
0000020 62 72 61 75 73 74 65 20 75 6e 64 20 72 61 75 73
0000030 63 68 74 65 20 73 63 68 6f 6e 20 77 69 65 64 65
0000040 72 20 72 65 63 68 74 20 6c 75 73 74 69 67 2c 20
0000050 64 65 72 20 53 63 68 6e 65 65 0d 0a 74 72 c3 b6
0000060 70 66 65 6c 74 65 20 65 6d 73 69 67 20 76 6f 6d
0000070 20 44 61 63 68 65 2c 20 64 69 65 20 53 70 65 72
0000080 6c 69 6e 67 65 20 7a 77 69 74 73 63 68 65 72 74
0000090 65 6e 20 75 6e 64 20 74 75 6d 6d 65 6c 74 65 6e
00000a0 20 73 69 63 68 20 64 61 7a 77 69 73 63 68 65 6e
00000b0 3b 20 0d 0a
00000b3
```

Der gelb unterlegte Ziffernblock enthält die hexadezimale Numerierung der abgebildeten ↗Bytes. Diese Zählung ist vom Programm od generiert

und nicht Teil der Datei selbst.[26] Die weiteren, jeweils zweiziffrigen hexadezimalen Zahlen repräsentieren den sinntragenden Inhalt der Datei. Welcher Text beim Öffnen der Datei mit einem Editor auf dem Bildschirm erscheint, hängt davon ab, welche Zeichentabelle (Codepage) verwendet wird. Der ↗Unicode-↗Codepage zufolge ist dem Zahlenwert x44 der „LATIN CAPITAL LETTER D“ zugeordnet. Da ↗Unicode den defacto-Standard in der aktuellen Computertechnologie darstellt, erscheint an Stelle der x44 auf quasi allen Rechnern ein großes „D“. Die Zahl x61 ist zugeordnet dem „LATIN SMALL LETTER A“ etc. Die Zahl x20 steht für das Leerzeichen. Grundsätzlich ist die Verwendung einer abweichenden ↗Codepage vorstellbar, in der beispielsweise die Zahl x44 nicht dem D, sondern einem beliebigen anderen Zeichen zugeordnet ist. Die resultierende Textdarstellung wäre dann natürlich vollkommen sinnlos.

Es ist augenfällig, dass die Zeilenumbrüche der numerischen Repräsentation von der des lesbaren Textes abweichen. Die Zeilenumbrüche des lesbaren Textes werden ebenso wie die Buchstaben mit Zahlen kodiert. Im vorliegenden Fall stehen die beiden Zahlen x0d und x0a gemeinsam für einen Zeilenumbruch. Leider ist die Kodierung von Zeilenumbrüchen bis heute – anders als die Kodierung von Schriftzeichen – nicht einheitlich geregelt (s. dazu S. 55, Kapitel „Kodierung des Zeilenendes„). Windows-Systeme schreiben für einen Zeilenumbruch die besagte Zahlenfolge x0dx0a in eine Datei, Unix-Systeme hingegen nur die Zahl x0a (http://de.wikipedia.org/wiki/_Zeilenumbruch). Öffnet man mit einem Unix-System eine mit Windows geschriebene Textdatei, so interpretiert Unix nur die Zahl x0a als Zeilenumbruch und betrachtet das x0d irrtümlich als wiederzugebendes Textzeichen. Den gängigen ↗Codepages zufolge ist x0d jedoch kein druckbares Zeichen, sondern steht für „carriage return“, also „Wagenrücklauf“ (in Anlehnung an Funktion und Terminologie einer Schreibmaschine). Dieses Zeichen wird dann mit der – sinnlosen und störenden – Zeichenfolge ^M (ctrl-M) im Text wiedergegeben.

- paste Das Kommando paste bewirkt die Zusammenfügung von zwei oder mehr Dateien in der Weise, dass die korrespondierenden ↗Zeilen jeder Datei zu jeweils einer ↗Zeile zusammengefügt werden, wobei zwischen den Ausgangszeilen jeweils ein Tabulator gesetzt wird.

[26] Die Zahl bezieht sich immer auf das erste ↗Byte der jeweiligen ↗Zeile.

Beispiel:

Inhalt einer Datei text1.txt:

```
Vorname1
Vorname2
```

Inhalt einer Datei text2.txt:

```
Nachname1
Nachname2
```

Das Kommando `paste text1.txt text2.txt` bewirkt folgende Ausgabe auf dem Bildschirm:

```
Vorname1            Nachname1
Vorname2            Nachname2
```

Anstelle eines Tabulators als Trenner zwischen den ursprünglichen ↗Zeilen lassen sich auch andere Zeichen wie z.B. „;“ oder „„ angeben (Option -d; Beispiel: `paste -d; text1.txt text2.txt`). paste bietet auch die Möglichkeit, das Ergebnis horizontal darzustellen (Option -s). Beispiel: `paste -d; -s text1.txt text2.txt`:

```
Vorname1;Vorname2
Nachname1;Nachname2
```

ACHTUNG: Speziell bei Verwendung der Optionen -d und -s liefert paste nur dann sinnvolle Ergebnisse, wenn die zu verbindenden Dateien im Unix-Dateiformat (s. oben S. 55, „Kodierung des Zeilenendes“) abgespeichert sind!

- sed sed ist ein nicht-interaktiver Stream-Editor. sed kann nichts, was awk nicht auch könnte. Sein großer Vorteil besteht darin, dass es wesentlich schneller arbeitet als awk (was allerdings im Rahmen der Korpuslinguistik so gut wie nie ins Gewicht fällt) und, wiederum verglichen mit awk, teilweise eine kompaktere Befehlssyntax besitzt (z.B. bei Ersetzungsvorgängen).

sed liest ebenso wie awk entweder von einer ↗Pipe oder aus einer oder mehreren ↗Eingabedateien. Die von sed auszuführenden Befehle können, wiederum wie bei awk, entweder auf der Kommandozeile angegeben oder in einer Skriptdatei abgelegt werden. sed arbeitet zeilenweise, wobei sed mitgeteilt werden muss, auf welche ↗Zeilen die

Befehle angewendet werden sollen. Diese sog. „Adressen“ können auf zweierlei Art angegeben werden: Entweder durch Angabe einer oder mehrerer Zeilennummern ($ steht jeweils für die letzte ↗Zeile) oder eines Zeilenbereichs (von-bis) oder durch Angabe eines Zeichenmusters, das in einer ↗Zeile auftreten muss.

sed gibt grundsätzlich die komplette Datei aus und *zusätzlich* die ↗Zeilen, die mit Adressen angesprochen sind, d.h. letztere erscheinen doppelt. Um dieses – sehr wahrscheinlich selten gewünschte – Verhalten zu verhindern, muss sed mit der Option -n aufgerufen werden.

sed besitzt eine überschaubare Anzahl an Editierbefehlen. Wir beschränken uns hier auf die Funktionen p und s. p steht für „print“ und bewirkt die Ausgabe einer ↗Zeile. s steht für „substitute“ und führt Ersetzungen von Zeichen und Zeichenketten durch.

sed verändert die Originaldatei nicht! Es schreibt lediglich das Ergebnis der Befehlsausführung auf die Standardausgabe (den Bildschirm) oder gegebenenfalls („>“!) in eine Datei.

Beispiel (Die Datei eichenorff2.txt enthalte folgenden Text):

```
Wem Gott will rechte Gunst erweisen,
Den schickt er in die weite Welt,
Dem will er seine Wunder weisen
In Berg und Wald und Strom und Feld.
Die Trägen, die zu Hause liegen,
Erquicket nicht das Morgenrot,
Sie wissen nur vom Kinderwiegen,
Von Sorgen, Last und Not um Brot.
Die Bächlein von den Bergen springen,
Die Lerchen schwirren hoch vor Lust,
Was sollt ich nicht mit ihnen singen
Aus voller Kehl und frischer Brust?
Den lieben Gott laß ich nur walten;
Der Bächlein, Lerchen, Wald und Feld
Und Erd und Himmel will erhalten,
Hat auch mein Sach aufs best bestellt!
```

Das Kommando

```
sed -n '/Morgenrot/p' eichendorff2.txt
```

bewirkt folgende Ausgabe:

```
Erquicket nicht das Morgenrot,
```

Die Zeichenfolge zwischen den Schrägstrichen (/) wird als Suchmuster interpretiert. ↗Zeilen, in denen dieses Muster auftritt, werden ausgegeben, was durch die Funktion „p“ unmittelbar hinter dem das Suchmuster abschließenden Schrägstrich veranlaßt wird. Die Option -n bewirkt, dass die Ausgabe des übrigen Textes unterdrückt wird.

```
sed -n '/Morgenrot/s/rot/blau/gp' eichendorff2.txt
```

Ausgabe:

```
Erquicket nicht das Morgenblau,
```

Die Funktion s/rot/blau/ gibt an, dass „rot“ durch „blau“ ersetzt werden soll. Das nachfolgende g (für „global“) bewirkt, dass, für den Fall, dass die Zeichenfolge „rot“ mehr als einmal in der betroffenen ↗Zeile vorkommt, *alle* Vorkommen ersetzt werden. Der Printbefehl „p“ wird unmittelbar dahinter geschrieben.

Die sed-Anweisung

```
sed 's/\([a-zäöüß][a-zäöüß]*\)\([\.,;!\?]\)/
          \1 \2/g' eichendorff2.txt
```

rückt alle Satzzeichen durch ein Spatium vom vorangegangenen Wort ab. \1 und \2 referenzieren die beiden Gruppen, die im Suchmuster durch die (mit \ maskierten) runden Klammern definiert wurden.

Da sed Zeilenweise operiert, ist das Entfernen von Zeilenumbrüchen aus einer Datei problematisch. Folgende, hier nicht weiter zu erläuternde Syntax erledigt diese Aufgabe:

```
sed ':a;N;$!ba;s/\n/ /g'
```

sed erlaubt Ersetzungen unter Verwendung der hexadezimalen Zahlencodes. Folgendes Kommando ersetzt den durch das Kommando „echo“ ausgegebenen Buchstaben „ä“ (xc3a4) durch die Buchstabenfolge ae:

```
echo ä | sed 's/\xc3\xa4/ae/g'
```

- sort Sortiert die ↗Zeilen einer Datei nach Maßgabe der den Buchstaben zugeordneten Zahlenwerte (nicht vergessen: In Dateien steht *nie* Text, sondern immer nur Zahlen!). Den Erwartungen

entsprechende Ergebnisse liefert das Kommando nur für Buchstaben, die im englischen Alphabet enthalten sind. Deutsche Umlaute, Sonderzeichen aller Art etc. folgen nach dem Buchstaben z. Ihre Reihenfolge orientiert sich an der Kodierung der Datei und der entsprechenden Reihenfolge der zugrundeliegenden Zeichentabelle (↗„Codepage“) . Zur Entfernung von Duplikaten in einer Datei (mit „Duplikat“ sind hier mehrfach in einer Datei vorkommende, identische *↗Zeilen* gemeint) kann man das Kommando sort mit der Option --unique (oder -u) aufrufen.

- split Mit split lassen sich sehr große Textdateien in gleich große Einzeldateien zerlegen, mit der Option „-l“ kann angegeben werden, wieviele ↗Zeilen jede Einzeldatei enthalten soll. Ohne spezielle Anweisung benennt split die Einzeldateien nach folgendem Muster: xaa, xab, xac, … , xba, xbc, … Das Kommando kann z.B. verwendet werden, um Dateien, deren Größe das Limit für den Import von Daten in eine ↗Datenbank über das PhpMyAdmin-Interface (s. dazu unten S. 152 A. 44) überschreitet, zu zerlegen und anschließend sukzessiv in die ↗Datenbank zu importieren.

- tail Das Kommando tail gibt standardmäßig die letzten zehn ↗Zeilen einer Datei aus. Als Option kann die Anzahl der auszugebenden ↗Zeilen explizit angegeben werden:

```
tail -25 textdatei.txt
```

- tr tr ersetzt *einzelne* Zeichen durch andere Zeichen.

  Beispiel:

```
tr a b < text1.txt
```

ersetzt alle „a“ in der Datei durch „b“. ACHTUNG: tr ersetzt keine Zeichenketten!

Beispiel:

```
tr ab cd < text1.txt
```

ersetzt *nicht* die Zeichenfolge „ab“ durch die Zeichenfolge „cd“, sondern jedes „a“ durch ein „c“ und jedes „b“ durch ein „d“.

ACHTUNG: Bei tr ist die Eingabeumlenkung < obligatorisch!

- wc „word count“ gibt die Anzahl von ↗Zeile, Wörtern (im Sinne von ↗Tokens!) und Zeichen in einer Datei aus. Das Kommando kann auch in ↗Pipes verwendet werden, etwa zur Feststellung der Anzahl von Dateien in einem Verzeichnis:

```
ls -l | wc
```

- wget wget ist ein Programm, mit dem man automatisch Daten aus dem Internet herunterladen und auf dem eigenen Rechner abspeichern kann.

Beispiel:

```
wget -m -p -k http://www.itg.uni-muenchen.de
```

wget liefert nicht auf jeder Webseite brauchbare Ergebnisse. Vor allem bei Webseiten, deren Inhalte ganz oder teilweise dynamisch generiert werden, treten Probleme auf. Häufig besteht die einzige Lösung darin, die Inhalte manuell herunterzuladen.

- xxd xxd ist hinsichtlich seiner Funktion weitgehend identisch mit od. Anders als jenes erlaubt es jedoch auch den Dump im binären Zahlenformat (xxd -b [dateiname]).

3.3.2.4 Klammer-Expansion

Die ↗Shell ist dazu in der Lage, Gruppen von Zeichen oder Zeichenketten, die jeweils von geschweiften Klammern umschlossen sind, systematisch miteinander zu kombinieren.

Beispiel:

```
echo {50,60,70}{w,m}{1,2,3}{1,2}.txt |
sed 's/ /\n/g'
```

Der Befehl echo bewirkt die Ausgabe auf dem Bildschirm. Die Inhalte in den geschweiften Klammern werden miteinander kombiniert, wobei die entstehenden Zeichenketten getrennt durch Leerzeichen ausgegeben werden. Um jede Zeichenkette in einer eigenen ↗Zeile zu erhalten, wird das Ergebnis an sed weitergegeben („gepipet“), das jedes Leerzeichen durch eine neue ↗Zeile ersetzt. Das Ergebnis sieht dann wie folgt aus:

```
50w11.txt
```

```
50w12.txt
50w21.txt
50w22.txt
50w31.txt
50w32.txt
50m11.txt
50m12.txt
50m21.txt
50m22.txt
50m31.txt
50m32.txt
… usw.
```

Verwendet man statt des Befehls „echo“ den Befehl „touch“, werden Dateien mit den erzeugten Zeichenketten als Dateinamen angelegt, mit „mkdir“ werden entsprechende Ordner erzeugt. Das Beispiel basiert auf dem oben vorgestellten Beispiel einer zeit- und personenbezogenen Systematik eines fiktiven Korpus. Man kann sich auf diese Weise eine Menge Tipparbeit ersparen und – was wichtiger ist – reduziert die Wahrscheinlichkeit von Tipp- oder systematischen Fehlern. Die mit „touch“ angelegten Dateien sind zunächst vollkommen leer und können/müssen dann mit Daten bzw. Text gefüllt werden.

3.3.2.5 Verknüpfung mehrerer Kommandos auf der Kommandozeile

In der Praxis ist es häufig von Vorteil, eine Vielzahl unterschiedlicher Kommandos zu bündeln. Im folgenden führen wir ein solches Verfahren anhand eines Beispiels vor.

Mehrere sed-Kommandos können bei einem einzigen Kommandoaufruf unmittelbar nacheinander ausgeführt werden. Im folgenden Beispiel erzeugt das Kommando „echo“ die Ausgabe einer Zeichenkette („hallo ...“). Diese wird durch den ↗Pipe-Operator (|) an das Kommando „sed“ weitergeleitet, das dann eine Reihe von Ersetzungsoperationen durchführt. Konkret werden alle Satzzeichen und Klammern durch Spatien von den Wörtern abgetrennt, so dass jedes Satzzeichen, jede Klammer und jedes Wort ein eigenes ↗Token bilden:

```
echo 'hallo! hallo? hallo, hallo; (hallo).' | sed 's/\./
./g; s/!/ !/g; s/\?/ ?/g; s/,/ ,/g; s/;/ ;/g; s/)/ )/g;
s/(/( /g;'
```

Will man diese Ersetzungen auf den Inhalt einer Text-Datei anwenden, schreibt man das sed-Kommando mit den Ersetzungsanweisungen, gefolgt von einem oder mehreren Dateinamen:

```
sed 's/\./ ./g; s/!/ !/g; s/\?/ ?/g; s/,/ ,/g; s/;/ ;/g;
s/)/ )/g; s/(/( /g;' datei1.txt datei2.txt *.csv
```

Das Ergbnis dieser Ersetzungen wird lediglich auf dem Bildschirm ausgegeben. Will man die Ersetzungen direkt in den Dateien selbst vornehmen, muss sed mit der Option „-i“ (= in place) aufgerufen werden.[27] VORSICHT: es erfolgt keine Rückfrage!

3.3.2.6 Job-Dateien

Es besteht außerdem die Möglichkeit, mehrere Unix-Kommandos zeilenweise in einer sog. Job-Datei abzulegen (Job-Dateien entsprechen in etwa den Batch-Dateien in der Windows-/DOS-Welt). Alle der dort abgelegten Befehle werden der Reihe nach von oben nach unten abgearbeitet. Die Datei darf nur mit einem Texteditor und auf keinen Fall etwa mit Word oder einem sonstigen Textverarbeitungsprogramm geschrieben werden!

Beispiel (Inhalt der job-Datei „test.job“):

```
/bin[28]/sort test.txt\
| sed 's/jetzt/now/g' \
> ersetzung.sed
```

Die Datei test.txt wird zunächst vom Kommando „sort“ sortiert und das Ergebnis dann an das Kommando „sed“ weitergeleitet; dieses ersetzt alle Vorkommen von „jetzt“ durch „now“. Das Ergebnis dieser Ersetzung wird anschließend in die Datei „ersetzung.sed“ geschrieben. Die Backslashes maskieren die Zeilenumbrüche (x0a), da die ↗Shell ansonsten die Ausführung der Kommandos aufgrund eines Syntaxfehlers verweigern würde.[29]

Der Aufruf der job-Datei erfolgt sodann auf der Kommandozeile einfach durch Angabe ihres Namens, wobei unbedingt der ↗Pfad zur Datei

---

[27] sed –i 's/\./ ./g; s/!/ !/g; s/\?/ ?/g; s/,/ ,/g; s/;/ ;/g; s/)/)/g; s/(/(/g;' datei1.txt datei2.txt *.csv

[28] Die Angabe des absoluten ↗Pfades ist bei Verwendung von Cygwin auf Windows-Rechnern wichtig, da es ein Windows-eigenes gleichnamiges Programm „sort“ gibt. Ohne die Angabe des absoluten Pfades in diesem Fall kann es passieren, dass statt des Cygwin-/Unix-Kommandos „sort“, das Windows-Programm ausgeführt wird, das anders funktioniert und andere Ergebnisse erzeugt.

[29] In diesem Fall ist außerdem wichtig, dass die Job-Datei im Unix-Dateiformat abgespeichert ist, da ansonsten (bei Windows-Dateiformat) am Zeilenende die Zeichenfolge 0d0a stehen würde, die Maskierung durch Backslash sich also auf 0d beziehen und 0a stehenbleiben würde.

dem Namen vorangestellt werden muss, da das System ansonsten vergeblich in den für Kommandos vorgesehenen Ordnern (im wesentlichen cygwin/bin) nach einem Kommando dieses Namens suchen würde. Sofern man mit cd bereits in das Verzeichnis gewechselt hat, in dem sich die job-Datei befindet, genügt die Angabe eines Punktes als Kürzel vor das aktuelle Verzeichnis:

```
./test.job
```

3.3.2.7 Shell-Scripts

Ein ↗Shell-Script kann man als eine besondere Art von Job-Datei bezeichnen, in der zusätzlich die umfangreichen Programmiermechanismen (Bedingungen, Schleifen) der ↗Shell verwendet werden können.

Beispiel:

```
1    #!/bin/sh
2
3    if [ -f Wortliste.txt ]
4    then
5      rm Wortliste.txt
6      echo "Datei Wortliste.txt geloescht."
7    fi
8
9    sed -f delete_unicodebom.sed zeichentabelle |
10   gawk -f genarab2beta.awk > arab2beta.sed
11
12   for file in *.txt
13   do
14     sed -f delete unicodebom.sed $file |
15     sed -f arab2beta.sed |
16     sed -f postbeta.sed >$file.beta
17     gawk -f makeliste.awk $file.beta >> Wortliste.txt
18   done
19
20   echo "Fertig."
```

Ein solches ↗Shell-Script muss in einer Datei abgespeichert werden, die dann durch Eingabe ihres Namens auf der Kommandozeile aufgerufen werden kann. Die Datei muss im Unix-Dateiformat abgespeichert werden (Kodierung des Zeilenendes mit x0a). Für den Aufruf muss der absolute oder relative ↗Pfad zur Datei mit angegeben werden. Im Fall, dass man sich im entsprechenden Verzeichnis befindet, muss dem Dateina-

men „./“ vorangestellt werden, wobei der Punkt für das aktuelle Verzeichnis steht.

3.3.2.8 „Pipen“ von Kommandos an die Shell

Besonders für die systematische Umbenennung von Dateien und/oder Ordnern ist die automatische Erzeugung entsprechender Kommando-Syntax von großem Nutzen. Der „Trick“ besteht darin, dass man die Liste eines Ordnerinhalts mit dem Kommando „ls“ ausgeben läßt und diese Liste anschließend mit einem oder mehreren Ersetzungskommandos (meist sed und/oder awk) so umbaut, dass daraus ein von der ↗Shell interpretierbares Umbenennungskommando entsteht. „Pipet“ man das Ergebnis dieser Umbau-Prozedur an die ↗Shell, so führt diese die entstandenen Kommandos aus. Im folgenden Beispiel werden in Dateinamen sämtliche Leerzeichen durch Unterstriche ersetzt.

```
ls | gawk '{printf "mv " gensub(/ /,"\\\\ ","g",$0)
"xxx"; gsub(/AIS /,"AIS"); print $0}' | gawk 'BEGIN
{FS="xxx";} {gsub(/ /,"-",$2); $2=gensub(/([0-9])-
/,"\\1_","g",$2); print $0}' | bash
```

Das Kommando ls gibt zunächst die Dateinamen aus, z.B. „AIS 0263 me ne vado.jpg“. Die beiden nachfolgenden gawk-Kommandos erzeugen folgende Zeichenkette:

```
mv AIS\ 0263\ me\ ne\ vado.jpg AIS0263_me-ne-vado.jpg
```

Dieser mv-Befehl wird anschließend an die ↗Shell (bash) „gepipet“, die ihn sodann ausführt. Dieses Verfahren ist bei einer größeren Anzahl von Dateien sehr sinnvoll.

3.4 Reguläre Ausdrücke

Unter „regulären Ausdrücken“ (kurz auch: RA, regex oder RE [engl.: Regular Expression]) versteht man ein Verfahren, mit dem man nahezu beliebige Zeichenfolgen beschreiben und suchen bzw. suchen und ersetzen kann. Die Suche mit Hilfe von RAs ist ein weit verbreitetes Konzept, das in vielen Textressourcen und Programmen Verwendung findet.[30] Sinnvoll sind RAs hauptsächlich dann, wenn man nach mehr als einem, exakt definierten Muster sucht. Wollte man beispielsweise in einem Text nach

[30] z.B. online-↗Datenbanken, Corpora etc.; vgl. etwa den "Tesoro della Lingua Italiana delle Origini" (http://tlio.ovi.cnr.it/TLIO/advric.htm)

dem Vorkommen der beiden Wörter „Haus“ und „Hans“ suchen, müßte man in den meisten herkömmlichen Textverarbeitungsprogrammen nacheinander zwei Suchen durchführen. Bei Verwendung von RAs ließe sich dies mit einer einzigen Suche bewerkstelligen:

Ha[un]s

Die eckigen Klammern zeigen an, dass es sich bei den darin eingeschlossenen Zeichen um Alternativen handelt, also: „entweder 'u' oder 'n'“.

Das Zeicheninventar regulärer Ausdrücke lässt sich zunächst in Literale und Metazeichen unterscheiden. Literale sind Zeichen, die für sich selbst stehen (a bezeichnet exakt das Zeichen a). Metazeichen sind Zeichen, die eine Sonderfunktion (z.B. Kontext- oder Wiederholungsoperatoren) besitzen. Elementare Metazeichen sind:

- **.** Der Punkt steht für ein beliebiges Zeichen (Zahl, Buchstabe, Spatium, Satzzeichen etc.)
- \ Der Backslash „maskiert“ Zeichen mit Sonderfunktion. Angenommen, man sucht nach mit einem Punkt abgeschlossenen Sätzen. Da der Punkt in den RA für ein beliebiges Zeichen steht, muss er maskiert werden.

  Beispiel:

  grep ' dar\.' findet die Zeichenkette „ dar“ gefolgt von einem Punkt.

Reguläre Ausdrücke stellen als Grundfunktionen „Angabe von Alternativen“ und „Angabe von Wiederholungen“ zur Verfügung. Die Kombination von beiden Funktionen ergibt nahezu grenzenlose Möglichkeiten.

3.4.1 Alternativen

Eine Möglichkeit der Angabe von Alternativen wurde bereits vorgestellt: die Verwendung von eckigen Klammern. Innerhalb der eckigen Klammern können, wie oben, entweder einzelne Zeichen oder Zeichenbereiche angegeben werden. Dabei bezeichnet ein Paar eckiger Klammern jeweils genau *ein* Zeichen aus dem angegebenen Zeichenrepertoire. Beispiel:

[134] vs. [0-9]

Der erste RA bezeichnet entweder eine 1 oder eine 3 oder eine 4. Der zweite RA bezeichnet eine Zeichenklasse, zu der alle Ziffern zwischen (und inklusive) 0 und 9 gehören.

Bei der Angabe von Zeichenbereichen mit Hilfe des Bereichszeichens „-“ ergibt sich die Menge der darin enthaltenen Zeichen aus der Reihenfolge der zugeordneten Zahlenwerte der ASCII-↗Tabelle (s. Anhang S. 199). Beispiel:

Die Zeichenklasse [MPY-b] sieht bei Verwendung der zugeordneten Zahlenwerte der ASCII-↗Tabelle folgendermaßen aus:

77	M
80	P
89	Y
90	Z
91	[
92	\
93	]
94	^
95	_
96	`
97	a
98	b

Anstelle von [MPY-b] könnte man also auch schreiben: [MPYZ[\]^_`ab] (ACHTUNG: in der Praxis funktioniert dieser RA nicht, da die eckigen Klammern und der Backslash innerhalb des RAs Sonderbedeutung besitzen und daher „maskiert“ werden müssen).

Zu beachten ist, dass die ASCII-↗Tabelle keine Zeichen enthält, die dem lateinischen bzw. englischen Alphabet fremd sind! Um beispielsweise deutsche Sonderzeichen in einen RA einzuschließen, müssen diese eigens hinzugefügt werden: [A-Za-zäöüß] bezeichnet einen beliebigen Buchstaben des *deutschen* Alphabets. Analoges gilt für Sonderzeichen anderer Alphabete wie etwa das französische oder portugiesische (Beispiel: [A-Za-zéèêâ]).

Weiteres Beispiel: Ein beliebiges Zeichen aus dem ASCII-Bereich zwischen dem Tabstop (x9) und der Tilde (x7e) wird mit dem RA [^\t-~] gefunden. Dieser RA kann nützlich sein, um in einem Text Zeichen aufzufinden, die *nicht* dem ASCII-Bereich angehören und daher bei automatischen Prozeduren prekär sein könnten. Die Negation des RAs im

Sinne von „eben *nicht* aus dem definierten Zeichenbereich“ wird durch die Voranstellung des „^“ innerhalb der eckigen Klammern bewirkt.

Alternativen können auch durch Verwendung des Operators „|“ angegeben werden:

Haus|Hof

findet „Haus“ oder „Hof“. ACHTUNG: Diese Syntax wird nicht von allen Programmen verstanden (ja: egrep, awk, sed[31]; nein: grep).

3.4.2 Wiederholungen

Für die Angabe von Wiederholungen verfügt das System der RAs mehrere Operatoren bzw. Methoden. Grundsätzlich gilt, dass Wiederholungsoperatoren unmittelbar hinter dem Zeichen (oder der Zeichenkette) gesetzt werden müssen, deren Wiederholung markiert werden soll. Dabei bedeuten:

?	kein oder genau ein Vorkommen
+	mindestens ein oder beliebig viele Wiederholungen
*	kein oder beliebig viele
{m}	exakt m Wiederholungen
{m,n}	mindestens m, maximal n Wiederholungen

Beispiele:

Za?hn	findet Zahn, aber auch Zhn
Ham+er	findet Hammer, Hamer aber auch Hammmmmmer
Ham*er	findet Hammer, Hamer, Hammmmmmer und Haer
Ham{2,3}er	findet Hammer und Hammmer

RAs gestatten auch die Definition von Gruppen mit Hilfe von runden Klammern (s.u. S. 86):

H(am){1,3}er	findet Hamer, Hamamer und Hamamamer

[31] sed muß in diesem Fall mit der Option –r aufgerufen werden. "-r" ermöglicht die Verwendung von "extended regular expressions".

Mit Hilfe von regulären Ausdrücken lassen sich auch Wiederholungen von Buchstaben, Wörtern und ganzen Wortgruppen auffinden (die RAs stehen zwischen Slashes: /RA/):

- Buchstabendoppelungen (mm, nn etc.): /`\([a-zA-Z]\)\1`/
- Wortwiederholungen (die die) [Achtung: Leerzeichen am Beginn!]: /`  \([^ ]\+\)\ s\+\1`/
- Wiederholung von Wortgruppen: /`\(.\+\)\ s\+\1`/

3.4.3 Kontextoperatoren

- ^ Das „Dach“ hat unterschiedliche Bedeutung: **Außerhalb** einer Zeichenklasse bezeichnet es den **Anfang** einer Zeichenkette. Damit kann, je nach gegebenem Kontext, der Anfang einer ↗Zeile, der Anfang eines Wortes, oder auch der Anfang eines Feldinhaltes in einer ↗Datenbank gemeint sein: /^Haus.*/ bezeichnet die Zeichenfolge „Haus“ am Beginn einer Zeile/eines Wortes/eines Datenfeldes.

  Das „Dach“ **innerhalb** einer Zeichenklasse (= Zeichenfolge innerhalb eckiger Klammern) bezeichnet, wie bereits erwähnt (s.o.), eine Negation: [^ab] findet alle Zeichen **außer** a und b.

- $ Das Dollarzeichen korrespondiert mit dem Anfangszeichen „^“ und bezeichnet das *Ende* einer Zeichenkette, also wiederum einer ↗Zeile, eines Wortes oder eines Datenfeldes. Der RA /.*heit$/ findet alle Zeichenketten am Ende einer ↗Zeile, eines Wortes, oder eines Datenfeldes, die auf -heit enden.

  Weitere Beispiele:

  Gegeben sei eine Textdatei mit der ↗Zeile „Der Weisheit letzter Schluss“ (Treffer werden mit „1“ markiert, keine Treffer mit „0“):

  grep '^Der.*' ⇒ 1

  grep '.*luss$' ⇒ 1

  grep '^letzter.*' ⇒ 0

awk '{print match($0,/^Der.*/)}' ⇒ 1

awk '{print match($3,/^l.*ter$/)}' ⇒ 1

3.4.4 Gruppierung und Rückreferenzierung

Durch die Verwendung von runden Klammern lassen sich Zeichenketten gruppieren. Bestimmte (nicht alle!) Programme, die die Verwendung von RAs erlauben, bieten die Möglichkeit, gruppierte Zeichenketten mit einer speziellen Syntax anzusprechen.

Beispiel (VIM):

Der VIM-Befehl:

```
:%s=\(.*\)\( Weisheit\)\(.*\)
\( Schluss\)=\1\3\2\4=g
```

erzeugt, angewandt auf die obige Beispieldatei ("Der Weisheit letzter Schluss"), folgende Ausgabe:

```
Der letzter Weisheit Schluss
```

Jedes Klammernpaar definiert eine Gruppe. Im Ersetzungsteil wird jede der definierten Gruppen durch eine Zahl angesprochen, der ein Backslash vorangestellt ist. \1 bezeichnet dabei die erste Gruppe, \2 die zweite usw. Man beachte: Auch die Spatien vor den Wörtern Weisheit und Schluss sind jeweils Teil der definierten Gruppen.

Je nach verwendetem Programm kann die Syntax leicht unterschiedlich sein. So muss im Editor VIM bei der Gruppenbildung jeder Klammer ein Backslash vorangestellt sein, im Ersetzungsteil hingegen darf den Zahlen jeweils nur ein Backslash vorangestellt werden. In der Programmiersprache AWK (s.u. S. 87) hingegen müssen – bei Verwendung der Ersetzungsfunktion „gensub“) – die runden Klammern **ohne** Backslash-Maskierung und die Rückreferenzen mit je zwei vorangestellten Backslashes geschrieben werden.

Beispiel (AWK-Syntax):

```
gensub(/(.*)(Weisheit)(.*)(Schluss)/,"\\1\\3
\\2\\4","g",$0)).
```

3.4.5 Abweichungen vom Standard

Ob und in welcher Weise RAs verwendet werden können, hängt stark von den eingesetzten Programmen ab. So stehen im Programm „grep" weniger Funktionen der RAs zur Verfügung als beispielsweise im Programm „egrep", bisweilen existieren Abweichungen bezüglich der Notwendigkeit zur Maskierung von (Sonder-)zeichen (bei Verwendung von VIM müssen z.B. die Wiederholungsoperatoren „+" und „?" jeweils mit einem Backslash maskiert werden, „*" hingegen nicht, etc.). Bei auftretenden Fehlern empfiehlt es sich zu prüfen, ob man eventuell gesuchte Sonderzeichen nicht maskiert hat, ob möglicherweise doppelte Maskierungen durch einfache zu ersetzen sind oder umgekehrt.

3.4.6 Dokumentationen und Links

- http://de.wikipedia.org/wiki/Regul%C3%A4rer_Ausdruck
- http://www.softpanorama.org/Editors/Vimorama/vim_regular_expressions.shtml (RAs in VIM)
- http://kitt.cl.uzh.ch/clab/regex/index.jsp (online-Workshop der Universität Zürich mit umfangreicher Dokumentation und Übungseinheiten)

3.5 Die Programmiersprache AWK

3.5.1 Grundlagen und Programmaufruf

Die Programmiersprache AWK[32] ist eine sogenannte „Skriptsprache"[33]. Sie ist vergleichsweise leicht erlernbar und sehr gut für die automatische Verarbeitung von Korpusmaterial geeignet. Aus Sicht der Korpuslinguistik ist sie vor allem unentbehrlich für die systematische Transformierung von Texten in ↗Tabellenformat, das wiederum Voraussetzung für den Import von Korpusmaterial in eine ↗Datenbank ist.

[32] Sofern die Programmiersprache AWK gemeint ist, wird der Name in Versalien geschrieben. Der Name des Kommandointerpreters wird hingegen klein geschrieben („awk").

[33] Skriptsprachen (auch „Interpreterprogramiersprachen") zeichnen sich innerhalb der Familie der Programmiersprachen u.a. durch eine relativ einfache „Grammatik" aus. Im Unterschied zu den „höheren" Programmiersprachen weisen sie eine gewisse Großzügigkeit im Hinblick auf Konventionen wie z.B. die sog. „Variablendeklaration" auf. Vgl. http://de.wikipedia.org/wiki/Skriptsprache. Weit verbreitete Skriptsprachen sind z.B. Perl, PHP und Python.

AWK-Skripte sind kleine Computerprogramme, die von einem geeigneten Kommandointerpreter (z.B. awk, gawk, nawk) ausgeführt werden müssen. Umfangreicher Programmcode wird normalerweise in eine Textdatei geschrieben. Zu diesem Zweck empfehlen wir die Verwendung des Editors „VIM“ (s.o. S. 35ff.).

Um das Programm zu starten, muss der Kommandointerpreter awk mit dem Namen der Skriptdatei als Parameter (wird hinter den Programmnamen geschrieben: awk **-f skript.awk**) auf der Cygwin-/Unix-Kommandozeile aufgerufen werden. Im Bereich der Korpuslinguistik wird awk stets zur Verarbeitung von Textdateien verwendet. Der oder die Namen der zu verarbeitenden Textdateien müssen beim Kommandoaufruf als weitere Parameter angegeben werden. Die zu verarbeitenden Dateien **müssen reine Textdateien sein**! Auf keinen Fall können Microsoft-Word-Dateien verarbeitet werden. Der Kommandoaufruf sieht wie folgt aus:

```
awk -f Skriptdatei.awk Textdatei_1.txt Textdatei_2.txt
... Textdatei_n.txt
```

Die Option -f steht für „file“ und bedeutet, dass awk das auszuführende Programm aus einer Datei lesen soll, nämlich der, deren Name unmittelbar folgt. Es ist empfehlenswert, die Skriptdateien stets mit der Extension awk zu versehen. Zum einen sind sie auf diese Weise gut von anderen Dateien unterscheidbar, zum anderen verwendet der Editor VIM dann automatisch das passende Farbschema, das die Syntax des Programmcodes paraphrasiert (sog. „Syntaxhighlighting“).

Kürzere Programmcodes können, umrahmt von Hochkommata ('), direkt in die Kommandozeile beim Programmaufruf geschrieben werden.

Beispiel:

```
awk '{print $0}' [Eingabedatei]
```

Der Kommandointerpreter AWK liegt in verschiedenen Varianten vor (awk, gawk, nawk etc.). Diese Varianten können sich im Detail hinsichtlich ihres Funktionsumfanges unterscheiden. Während bestimmte zentrale Befehle von allen Interpretern „verstanden“ werden, gibt es einige wenige, zumeist jüngere Kommandos (z.B. die Sortierfunktion „asort“ zur Sortierung von Zahlen und Text), die *nicht* von allen Interpretern verstanden werden. So kann es passieren, dass ein Programmcode, der

vom Interpreter gawk problemlos ausgeführt wird, bei Verwendung von awk zu einer Fehlermeldung führt. **Wir empfehlen grundsätzlich die Verwendung des Interpreters „gawk“** (→ http://www.gnu.org/software/gawk). Alle im folgenden gegebenen Erläuterungen und Beispiele beziehen sich auf die Verwendung von gawk.[34]

3.5.2 Aufbau eines AWK-Skriptes

Ein AWK-Skript kann grundsätzlich in drei Programmsektionen eingeteilt werden, wobei mindestens eine der Sektionen in einem Skript vorhanden sein muss:[35]

```
BEGIN {
  BEGIN-Sektion
}
{
  HAUPT-Sektion
}
END {
  END-Sektion
}
```

Für die einzelnen Sektionen gilt: Die Programmieranweisungen der BEGIN- bzw. END-Sektion werden ausgeführt, bevor eine ↗Zeile bzw. nachdem alle ↗Zeilen einer Input-Datei gelesen wurden. Das Programm der HAUPT-Sektion wird jeweils vollständig für jede einzelne ↗Zeile der Input-Datei ausgeführt.

3.5.3 Funktionsweise im Detail

Nach dem Kommandoaufruf öffnet awk zunächst die Skriptdatei und prüft deren Inhalt hinsichtlich Konformität (Prüfung, ob der Code den Regeln der Sprache entspricht; sog. „Parsen“). Sollte die Datei Syntaxfehler enthalten, wird die Bearbeitung abgebrochen und eine entsprechende Fehlermeldung ausgegeben. Hier ein Beispiel: Das folgende Skript:

```
{
     print "Hallo!;
}
```

[34] Für einen ersten Überblick sind die Seiten http://de.wikipedia.org/wiki/Awk und http://de.wikibooks.org/wiki/Awk empfehlenswert. Detailliertere Informationen speziell zum Einsatz von gawk findet man unter https://www.gnu.org/software/gawk/manual/gawk.html.

[35] Die alleinige Verwendung einer END-Sektion ist sinnlos und kann nicht verwendet werden.

bewirkt folgende Fehlermeldung auf dem Bildschirm:

```
$ gawk -f skript.awk
gawk: skript.awk:2:  print "Hallo!;
gawk: skript.awk:2:        ^ Nicht-beendeter ↗String
```

Ursache dieser Fehlermeldung ist, dass im Skript die vor „Hallo!“ stehenden Anführungszeichen nicht geschlossen wurden. gawk zeigt außerdem an, in welcher ↗Zeile der Skriptdatei sich der Fehler befindet (in diesem Fall in der ↗Zeile 2). Das fehlerfreie Skript würde so aussehen:

```
{
     print "Hallo!";
}
```

und folgende Bildschirmausgabe erzeugen:

```
$ Hallo!
```

Wenn die Syntaxprüfung der Skriptdatei keinen Fehler ergeben hat, wird mit der Ausführung des Programmcodes begonnen. Sofern das Skript keinen sog. BEGIN-Block enthält (mehr dazu weiter unten), öffnet gawk die erste der beim Programmaufruf angegebenen Textdateien und liest deren Inhalt **zeilenweise**, beginnend bei der ersten bis zur letzten ↗Zeile. Mit dem Einlesen jeder neuen ↗Zeile „vergißt“ gawk die vorangegangene ↗Zeile. Nach dem Schließen der letzten ↗Zeile der ↗Eingabedatei schließt gawk die ganze Datei und springt – sofern angegeben – zur nächsten der beim Kommandoaufruf angegebenen ↗Eingabedateien. Wenn die letzte ↗Zeile der letzten ↗Eingabedatei und auch die ↗Eingabedatei selbst geschlossen ist, beendet gawk seine Arbeit (sofern das Skript keinen sog. END-Block enthält; dazu weiter unten mehr). Anschließend springt der ↗Cursor in eine neue Eingabezeile.

Da der Programmcode des Skripts von gawk beim Durchlaufen der ↗Eingabedateien auf jede ↗Zeile angewandt wird, ist es sehr hilfreich, wenn alle ↗Zeilen strukturell identischen Inhalt besitzen. Naheliegend wäre z.B., grundsätzlich alle Sätze eines Textes in jeweils eigene ↗Zeilen zu schreiben.

Sämtlicher „Output“, der von gawk erzeugt wird, erscheint standardmäßig auf dem Bildschirm (Standardauagabe oder „stdout“). Wie bei allen Unix-Kommandozeilen-Tools läßt sich dieser Output durch die

Operatoren „>“ oder „|“ in Dateien umleiten bzw. an weitere Kommandos „pipen“.

3.5.3.1 Konkretes Beispiel

Gegeben sei die Datei tabelle.csv mit folgendem Inhalt:

```
Bayern     -  Nürnberg   3 : 0 (2 : 0)  69000 Zuschauer
Frankfurt  -  Leverkusen 2 : 1 (0 : 0)  47600 Zuschauer
Stuttgart  -  Hannover   0 : 2 (0 : 1)  49000 Zuschauer
Schalke    -  Karlsruhe  0 : 2 (0 : 0)  61482 Zuschauer
Hertha     -  Cottbus    0 : 0 (0 : 0)  48719 Zuschauer
Bielefeld  -  Hamburg    0 : 1 (0 : 0)  22800 Zuschauer
Wolfsburg  -  Rostock    1 : 0 (0 : 0)  26127 Zuschauer
Duisburg   -  Bremen     1 : 3 (1 : 1)  31006 Zuschauer
Dortmund   -  Bochum     2 : 1 (1 : 1)  72200 Zuschauer
```

Auf diese ↗Eingabedatei soll folgendes Skript mit dem Dateinamen „skript.awk“ angewendet werden (in diesem Zusammhang nicht sinnvoll, aber möglich):

```
{
  print "Hallo!";
}
```

Das Kommando `print` bewirkt eine Ausgabe auf dem Bildschirm. Der ↗String, der ausgegeben werden soll, steht hinter dem Kommando in doppelten Anführungsstrichen.

Anweisungen bzw. Blöcke zusammengehörender Anweisungen müssen stets von geschweiften Klammern umschlossen sein. Gewöhnen Sie sich an, die Struktur eines Skripts durch Einrückungen zu verdeutlichen und jedes Kommando mit einem Strichpunkt abzuschließen. (Sie sind nicht zwingend erforderlich, jedoch in vielen anderen Programmiersprachen üblich und dort oft obligatorisch [z.B. PHP].)

Der Kommandoaufruf „`gawk -f skript.awk tabelle.txt`“ erzeugt folgenden Output:

```
Hallo!
Hallo!
Hallo!
Hallo!
Hallo!
Hallo!
Hallo!
Hallo!
```

```
Hallo!
```

Das Kommando print „Hallo!“ wird für *jede ↗Zeile* der ↗Eingabedatei ausgeführt. Da die ↗Eingabedatei insgesamt 9 ↗Zeilen umfaßt, wird 9-mal „Hallo!“ auf den Bildschirm geschrieben.

Beim Einlesen jeder ↗Zeile der ↗Eingabedatei zerlegt gawk diese in einzelne Felder. Es orientiert sich dabei in der Grundeinstellung an den sog. Whitespace-Charakters, also Leerzeichen („blank“, „space“) und Tabulatorzeichen: Ein oder mehrere aufeinanderfolgende dieser Whitespace-Charakters werden von gawk jeweils als ↗Feldtrenner (abgelegt in der ↗Variablen FS [= Field ↗Separator]) interpretiert. Die Feldeinteilung kann gezielt verändert werden, indem der ↗Feldtrenner umdefiniert wird. So kann es z.B. sinnvoll sein, statt der White-Space-Charakters Kommata oder Semikola als ↗Feldtrenner zu betrachten.

Die Zeichenfolgen zwischen den ↗Feldtrennern sind schließlich die Felder, deren Inhalt von gawk in speziellen ↗Variablen abgelegt wird:

$1 ⇒ Inhalt des ersten Feldes

$2 ⇒ Inhalt des zweiten Feldes

usw.

Die Anzahl der Felder einer ↗Zeile speichert gawk in der ↗Variablen NF („number of fields“). Das letzte Feld einer jeden ↗Zeile ist folgendermaßen ansprechbar:

$NF ⇒ Inhalt des letzten Feldes

Die Nummer der jeweils gerade gelesenen ↗Zeile der ↗Eingabedatei wird in der ↗Variablen NR abgelegt, der Name der gerade verarbeiteten ↗Eingabedatei in der ↗Variablen FILENAME.

Der komplette Inhalt der jeweils gerade gelesenen ↗Zeile der ↗Eingabedatei wird in der ↗Variablen $0 abgelegt.

Übersicht:

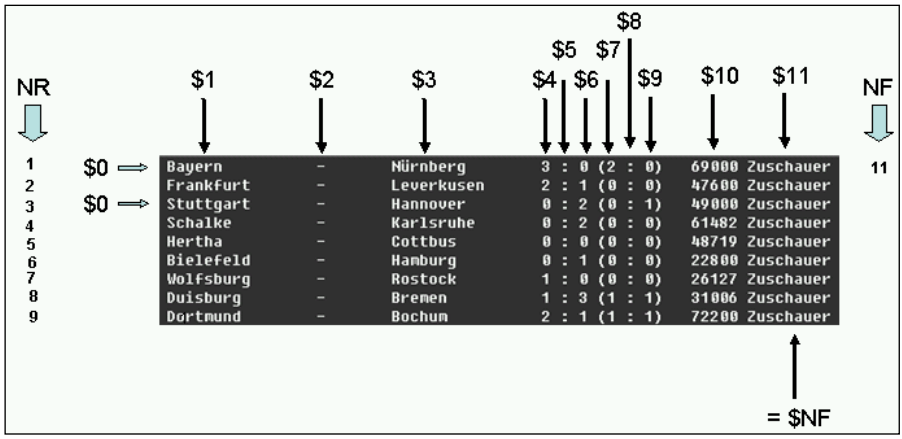


Abbildung 18: Die Datensicht von AWK – Einteilung in ↗Zeilen (records) und Felder (fields)

Der Inhalt jeder ↗Variablen läßt sich durch das Kommando „print“ ausgeben. Während „wörtlich“ auszugebende Zeichenketten wie etwa „Hallo!“ von Anführungszeichen umschlossen sein müssen, darf genau das bei Namen von ↗Variablen *nicht* der Fall sein (also $0, nicht „$0“). Das Setzen eines Kommas außerhalb von Anführungszeichen bewirkt die Ausgabe eines ↗Feldtrenners (in der Grundeinstellung ein Leerzeichen; abgelegt in der ↗Variablen OFS [= Output Field ↗Separator]).

Beispiel:

```
{
  print NR ". Das Spiel zwischen" , $1 , "und" , $3 ,
    "haben" , $10 , $NF , "gesehen";
}
Ausgabe:

1. Das Spiel zwischen Bayern und Nürnberg haben
     69000 Zuschauer gesehen
2. Das Spiel zwischen Frankfurt und Leverkusen haben
     47600 Zuschauer gesehen
3. Das Spiel zwischen Stuttgart und Hannover haben 49000
     Zuschauer gesehen
4. Das Spiel zwischen Schalke und Karlsruhe haben 61482
     Zuschauer gesehen
5. Das Spiel zwischen Hertha und Cottbus haben 48719
     Zuschauer gesehen
```

```
6. Das Spiel zwischen Bielefeld und Hamburg haben 22800
     Zuschauer gesehen
7. Das Spiel zwischen Wolfsburg und Rostock haben 26127
     Zuschauer gesehen
8. Das Spiel zwischen Duisburg und Bremen haben 31006
     Zuschauer gesehen
9. Das Spiel zwischen Dortmund und Bochum haben 72200
     Zuschauer gesehen
```

3.5.3.2 Benutzerdefinierte Variablen

Neben den bereits vorgestellten Systemvariablen wie $0, $1, NR usw. gibt es die Möglichkeit, benutzerdefinierte ↗Variablen zu verwenden. Man unterscheidet zwei Typen:

- „skalare“ ↗Variablen
- „indizierte“ ↗Variablen, sog. ↗Arrays

Skalare Variablen werden ganz einfach durch die Vergabe eines Namens (Achtung: keine Sonderzeichen, keine Leerzeichen!) und durch die Zuweisung eines Wertes (Zahlen und/oder Buchstaben) mit Hilfe des Zuweisungsoperators „=“ angelegt, d.h. zugleich deklariert und mit einem Wert besetzt. Beispiel:

```
meine_telefonnummer="0122/123456"
```

Das Kommando

```
print meine_telefonnummer
```

hätte folgendes Ergebnis:

```
0122/123456
```

↗Arrays sind ↗Variable mit „Unteradressen“. Der Name eines ↗Arrays kann ebenfalls frei gewählt werden (wiederum Achtung: keine Sonderzeichen, keine Leerzeichen!). Die Unteradressen werden zwischen eckigen Klammern unmittelbar (kein Leerzeichen!) hinter den Namen des ↗Arrays geschrieben. Die Wertzuweisung erfolgt auch hier mit dem Operator „=“. Beispiel:

```
meine_buecher[1]="Homer, Ilias"
meine_buecher[2]="Süßkind, Das Parfum"
meine_buecher[5]="Auguste Lechner, Parzival"
```

Die „Unteradressen“ sollten (wegen des dann leichter realisierbaren „Durchlaufens“ etwa im Rahmen von Schleifen, s. unten 3.5.3.4) Zahlen sein, könnten jedoch auch aus Buchstaben(folgen) bestehen:

```
meine_verwandten["bruder"]="Kain"
```

Wie oben gezeigt, müssen die Zahlen der „Unteradressen“ nicht fortlaufend sein.

Das Kommando

```
print meine_buecher[2];
```

hätte folgendes Ergebnis:

```
Süßkind, Das Parfum
```

Das Kommando

```
print meine_buecher[3];
```

würde gar keine Ausgabe bzw. eine Ausgabe ohne Inhalt bewirken (lediglich eine Leerzeile).

Beispiel – Belegung und Ausgabe von Variablenwerten:

```
BEGIN {
  OFS=" ### "; #36    (1)
  print OFS; #        (2)
}
{
  print FILENAME; # (3)
  print NR; #       (4)
}
```

Anmerkungen:
(1) Zuweisung der Zeichenfolge " ### " an die Variable OFS[37].
(2) Ausgabe des Inhalts der ↗Variablen OFS, in diesem Fall also " ### ".
(3) Ausgabe des Namens der aktuell gelesenen ↗Eingabedatei.

[36] Ein Gitterkreuz (auch: "Hash") dient in einem AWK-Skript als Kommentarzeichen. Sämtlicher Text, der rechts von einem Gitterkreuz bis zum Zeilenende steht, wird als Kommentar betrachtet und bei Ausführung des Programms ignoriert.
[37] s. oben S. 93.

(4) Ausgabe der Nummer der aktuell verarbeiteten ↗Zeile der aktuell gelesenen ↗Eingabedatei.

Beispiel: **Zählen** (inkrementieren, dekrementieren) mit ↗Variablen:

```
BEGIN {
k=0; #                          (1)
zahl_1=0; #                     (2)
zahl_2=0; #                     (3)
zahl_3=5; #                     (4)
zahl_4=5; #                     (5)
 while (++k <=5) { #            (6)
  print "Durchlauf " k;
  print "Zahl_1: " zahl_1++; # (7)
  print "Zahl_2: " ++zahl_2; # (8)
  print "Zahl_3: " zahl_3--; # (9)
  print "Zahl_4: " --zahl_4; # (10)
 }
}
```

Anmerkungen:
(1) Zuweisung des Wertes 0 an die ↗Variable k;
(2) Zuweisung des Wertes 0 an die ↗Variable zahl_1;
(3) Zuweisung des Wertes 0 an die ↗Variable zahl_2;
(4) Zuweisung des Wertes 5 an die ↗Variable zahl_3;
(5) Zuweisung des Wertes 5 an die ↗Variable zahl_4;
(6) Solange k einen Wert kleiner oder gleich 5 hat, führe die folgenden Befehle aus; bereits *vor* dem ersten Durchlauf erhält k den Wert 1 (++k)
(7) Beim ersten Durchlauf hat Zahl_1 den Wert 0!
(8) Beim ersten Durchlauf hat Zahl_2 den Wert 1!
(9) Beim ersten Durchlauf hat Zahl_3 den Wert 5!
(10) Beim ersten Durchlauf hat Zahl_4 den Wert 4!

3.5.3.3 Bedingungen

Häufig ist es erforderlich, die Ausführung von Befehlen an Bedingungen zu knüpfen. Zu diesem Zweck kennt AWK die if-Konstruktion.

Beispiel:

```
{
if ($0 ~ /Hannover/) {
print NR ":" , $0;
  }
}
```

Ausgabe:

```
3: Stuttgart  - Hannover 0:2  (0:1)   49000   Zuschauer
```

Die Bedingung lautet, dass die aktuell gelesene ↗Zeile ($0) nur dann ausgegeben werden soll, wenn sie die Zeichenfolge „Hannover“ enthält.

Der Operator „~“ (Tilde) bedeutet „match“, anders ausgedrückt: Wenn die ↗Zeile die Zeichenfolge **enthält**. „match“ erlaubt die Verwendung von *Regulären Ausdrücken* (s. S. 81). Als Begrenzer der Zeichenfolge werden Schrägstriche (statt Anführungszeichen) verwendet.

Gleichheit wird mit dem Operator „==“ (doppeltes Gleichheitszeichen) geprüft. In diesem Fall ist die Verwendung von *Regulären Ausdrücken* *nicht* möglich.

Weiteres Beispiel:

```
{
 if (match($0,/[A-ZÄÖU][a-zäöüß]+heit/)) { #       (1)
  print FILENAME, NR, substr($0,RSTART,RLENGTH); # (2)
 }
}
```

Anmerkungen:
(1) Wenn $0 den regulären Ausdruck enthält ...
(2) Ausgabe der von match gefundenen Zeichenkette mit Angabe von Dateinamen und Zeilennummer.

Es ist auch möglich, mehrere Bedingungen miteinander zu verknüpfen. Diesem Zweck dienen die Operatoren „&&“ (logisches „Und“) und „||“ (logisches „Oder“). Zur Erläuterung s. dazu das AWK-Skript im Anhang (S. 209).

3.5.3.4 Schleifen

Sehr wichtig ist die Möglichkeit, sogenannte „Schleifen“ zu programmieren. Eine Schleife veranlaßt die Ausführung von einem oder mehreren Befehlen für eine exakt definierte Anzahl von Wiederholungen. Die Syntax sieht wie folgt aus:

```
{
for (Zahl=1;Zahl<=NF;Zahl=Zahl+1) {
print NR, Zahl, $Zahl;
  }
```

```
}
```

Ausgabe:

```
1    1    Bayern
1    2    -
1    3    Nürnberg
1    4    3
1    5    :
1    6    0
1    7    (2
1    8    :
1    9    0)
1    10   69000
1    11   Zuschauer
2    1    Frankfurt
2    2    -
2    3    Leverkusen
[... etc. ...]
```

Die Schleife wird eingeleitet mit dem Schlüsselwort „for“. Die Schleifendefinition erfolgt unmittelbar dahinter in runden Klammern. Die Definition ist durch Semikola (;) dreigeteilt. Im ersten Teil erfolgt die Initialisierung der ↗Variablen „Zahl“, wobei die Bezeichnung willkürlich ist (häufig wird der Name „i“gewählt, weil es sich um eine Ganzzahl (also keinen Bruch o.ä.) handelt und die englische Bezeichnung dafür „integer“ lautet). Durch den Operator „=“ wird in diesem ersten Teil eine ↗Variable namens „Zahl“ eingeführt und dieser gleichzeitig der Wert „1“ zugewiesen. Der dritte (letzte) Teil der Definition enthält die Vorschrift, was mit der ↗Variablen „Zahl“ nach jedem Schleifendurchlauf geschehen soll. Zahl=Zahl+1 (mögliche Kurzschreibweise: Zahl++) bedeutet, dass der jeweils in „Zahl“ abgelegte Wert genau um den Wert 1 erhöht werden soll. Der mittlere Teil der Schleifendefinition schließlich enthält die sog. „Laufzeitbedingung“: Die Schleife soll nur so lange durchlaufen werden, wie die in diesem Teil definierte Bedingung erfüllt ist. Im konkreten Fall muss der Wert von „Zahl“ stets kleiner oder höchstens gleich dem Wert sein, der in der ↗Variablen NF enthalten ist. NF aber enthält jeweils die Anzahl der Felder in einer ↗Zeile (s.o.). Das bedeutet, dass die Schleife mit dem Wert 1 beginnt und solange durchgeführt wird, bis Zahl den Wert von NF, im konkreten Fall also 11 erreicht hat. In diesem Moment wird die Abarbeitung der Schleife abgebrochen.

Die Anweisung, die bei jedem Schleifendurchlauf ausgeführt werden soll, steht hinter der Schleifendefinition in geschweiften Klammern:

print NR, Zahl, $Zahl; – Es handelt sich um eine schlichte Ausgabeanweisung. Verglichen mit den vorangegangenen Beispielen ist neu, dass hier gleichsam zwei ↗Variable miteinander kombiniert sind: Das Dollarzeichen spricht jeweils die Felder einer ↗Zeile an. Da die ↗Variable „Zahl“ beim Durchlauf der Schleife nacheinander die Werte von 1 bis 11 (NF) annimmt, lautet die Anweisung nacheinander (im Fall der ↗Zeile 9, so dass NR den Wert „9“ enthält): „print 9, 1, $1;“ ⇒ „print 9, 2, $2;“ ⇒ ... ⇒ „print 9, 11, $11;“ (nur zur Veranschaulichung; die Syntax ist nicht ganz korrekt, da die Werte von NR [9] und Zahl [11] so nicht in den Programmcode geschrieben werden dürfen). Da das Kommando „print“ hinter die ausgegebene Zeichenfolge stets einen Zeilenvorschub schreibt, erscheinen die Felder jeder ↗Zeile nun untereinander in je eigenen ↗Zeilen. Durch das Voranstellen der ↗Variablen NR und Zahl steht vor jedem Feldinhalt die Nummer der ↗Zeile und die Nummer des Feldes innerhalb der ↗Zeile.

Weitere Beispiele:

- Zählen von 5 bis 9:

```
BEGIN {
  for (zahl=5; zahl<=9; zahl++) { #    (1)
    print zahl; #                      (2)
  }
}
```

Anmerkungen:
(1) Die Schleife beginnt mit dem Wert 5 für Zahl; bei jedem Schleifendurchlauf wird der Wert von Zahl um den Wert 1 erhöht (Operator "++"!).
(2) Nacheinander werden die Werte 5,6,7,8 und 9 ausgegeben.

- Retrograde Ausgabe der ↗Zeile 5 einer beliebigen Textdatei:

```
{
  if (NR==5) {
    for (i=NF; i>=1;i--) {
      print $i;
    }
  }
}
```

3.5.3.5 Operatoren

Zuweisungsoperator: **=** (weist einer ↗Variablen einen Wert zu)

Vergleichsoperatoren: **==** (Identität), **!=** (ungleich), **~** ("match"; Verwendung von regulären Ausdrücken), **!~** ("does not match")

mathematische Operatoren: **+** (Addition), **-** (Subtraktion), ***** (Multiplikation), **/** (Division), **%** („Modulo"; ganzzahliger Rest einer Division)

Für die automatische Addition bzw. Subtraktion der Zahl 1 zu bzw. von einem gegebenen Wert gibt es die Operatoren ++ bzw. --, wobei folgender Unterschied zu beachten ist: zahl++ addiert den Wert 1 **nach** Aufruf der ↗Variablen, ++zahl addiert den Wert 1 **vor** dem Aufruf der ↗Variablen (zur Illustration s. das Beispiel oben S. 96).

3.5.3.6 Kommandos

Kommandos weisen AWK an, bestimmte Aktionen auszuführen. Die wichtigsten sind:

print ⇒ Ausgabe auf Bildschirm, gefolgt von Zeilenumbruch

printf ⇒ Ausgabe auf Bildschirm, *ohne* Zeilenumbruch

getline ⇒ Lesen aus einer Datei (selten gebraucht, daher hier nicht ausführlich beschrieben)

3.5.3.7 Funktionen

↗Funktionen arbeiten alle nach demselben Prinzip: Sie übernehmen Zeichen (Zahlen und/oder Buchstaben; sog. „Argumente"), verarbeiten sie und geben als Ergebnis Zeichen (wiederum Zahlen und/oder Buchstaben; sog. ↗Rückgabewert) zurück. Die Argumente werden, getrennt durch Kommata, grundsätzlich hinter den Namen der ↗Funktion zwischen runde Klammern geschrieben. AWK kennt eine Reihe von vordefinierten Funktionen. Wir beschränken uns auf die von Korpuslinguisten am häufigsten benötigten. Aus Gründen der Ökonomie werden in den Beispielen stets „feste" Zeichenketten als Argumente verwendet. In produktiven Skripts werden an dieser Stelle stets ↗Variable ($0, $1, $NR etc.) verwendet:[38]

- **gensub(regulärer_Ausdruck, Ersetzung, Treffer, Ziel)**

[38] Für eine sehr gute, vollständige Dokumentation sei hier auf das entsprechende Kapitel des AWK-Handbuchs des GNU-Projekts (https://de.wikipedia.org/wiki/GNU-Projekt) verwiesen: https://www.gnu.org/software/gawk/manual/html_node/String-Functions.html.

Die ↗Funktion gensub() führt Ersetzungen durch und benötigt vier Argumente:

- Die Zeichenfolge, nach der gesucht werden soll
- Die Zeichenfolge, die für die gesuchte eingesetzt werden soll
- Nummer des Treffers innerhalb der zu suchenden Zeichenfolge (für den Fall, dass das gesuchte Muster öfter als einmal gefunden wird; „g“ steht für *alle* [„global“] Treffer)
- Die Zeichenfolge, in der gesucht und ersetzt werden soll.

```
gensub(/Montag/,"Dienstag","g","Heute ist Montag");
```

Der ↗Rückgabewert sähe in diesem Fall folgendermaßen aus:

```
Heute ist Dienstag
```

Um sich den ↗Rückgabewert ausgeben zu lassen, muss man den Funktionsaufruf mit einem Print-Kommando verbinden:

```
printf gensub(/Montag/,"Dienstag","g","Heute ist
Montag");
```

- **gsub(regulärer_Ausdruck, Ersetzung, Ziel)**

Die Funktion gsub führt ebenfalls Ersetzungen durch.

Beispiel:

```
gsub(/Montag/,"Dienstag","Heute ist Montag");
```

Ein wesentlicher Unterschied zu gensub() ist der ↗Rückgabewert von gsub(), der lediglich die Anzahl der erfolgten Ersetzungen wiedergibt:

```
printf gsub(/Montag/,"Dienstag","Heute ist Montag");
```

ergibt die Zahl „1“, da genau eine Ersetzung vorgenommen wurde. Ein weiteres Beispiel: Der ↗Rückgabewert dieser Ersetzung:

```
printf gsub(/und/,"oder","Apfel und Birne und Orange");
```

wäre „2“, da die Zeichenfolge „und“ im Ziel zweimal ersetzt wurde.

Ein weiterer Unterschied besteht darin, dass die Anwendung dieser Funktion nur auf ↗Variablen sinnvoll ist, da das Ersetzungsergebnis sonst nicht angesprochen werden kann. Daher ist folgendes Vorgehen zwingend nötig:

```
zeichenkette="Apfel und Birne und Orange";
printf gsub(/und/,"oder",zeichenkette);
printf zeichenkette;
```

Ergebnis:

```
2 Apfel oder Birne oder Orange
```

gsub kann auch mit nur zwei Argumenten verwendet werden:

```
gsub(/xy/,"ab");
```

In diesem Fall wird die Ersetzung in der kompletten gerade gelesenen Eingabezeile (= $0) durchgeführt.

- **split(Zeichenkette, ↗Array, Separator)**

Die Funktion split zerlegt eine beliebige Zeichenkette nach Maßgabe eines frei definierbaren Feldtrennungszeichens in Einzelteile, die in einem ↗Array abgelegt werden.

```
Rückgabewert=split(ziel,name_des_arrays,Feldtrenner);
n=split("Apfel oder Birne oder Orange",fruechte,"oder ");
```

Der ↗Rückgabewert ist die Anzahl der Felder des generierten ↗Arrays.

```
printf split("Apfel oder Birne oder Orange",fruechte," oder ");
```

ergäbe also den Wert 3, da fruechte[1] = Apfel, fruechte[2] = Birne und fruechte[3] = Orange, insgesamt also drei Elemente vorhanden sind.

```
printf fruechte[1];
```

ergäbe „Apfel“;

- **substr(Zeichenkette,Start,laenge)**

substr() liefert als ↗Rückgabewert einen Ausschnitt aus der als erstem Parameter übergegeben Zeichenkette. Dieser Ausschnitt beginnt bei der

mit „start“ angegebenen Buchstabenposition und ist so viele Zeichen lang, wie in „laenge“ angegeben. Beispiel:

```
print substr("Apfel und Birne und Orange",5,13)
```

Ergebnis:

```
l und Birne u
```

Wiederum kann anstelle einer Zeichenkette eine Systemvariable übergeben werden (vgl. unten unter „match“). Besonders sinnvoll ist der Einsatz von substr, um mit Hilfe von regulären Ausdrücken gefundene Zeichenketten weiterzuverarbeiten (zu speichern, zu drucken; vgl. ebenfalls unten zu „match“).

- **match(Zeichenkette, Regulärer_Ausdruck)**

match() prüft auf das Vorhandensein einer Zeichenkette in einer anderen Zeichenkette (die gesuchte Zeichenkette ist der *zweite* Parameter):

```
printf match("Apfel und Birne und Orange", /und/);
```

Dieses Kommando liefert (überraschenderweise) den Wert „7“, nämlich die Position des ersten Zeichens der gefundenen Zeichenkette. match() setzt außerdem den Wert der ↗Variablen „RSTART“ (für „Start des Regulären Ausdrucks“) und „RLENGTH“ (für „Länge des regulären Ausdrucks“). Der Wert von RSTART ist also identisch mit dem ↗Rückgabewert. Nützlich ist dies für die Verwendung der Funktion substr(), die das Ausschneiden der gefundenen Zeichenkette erlaubt:

```
zeichenkette="Apfel und Birne und Orange";
match ("Apfel und Birne und Orange", /A.*n/);
printf substr(zeichenkette,RSTART,RLENGTH);
```

Das Ergebnis wäre:

```
Apfel und Birne und Oran
```

Man sieht, dass match die längstmögliche Zeichenkette, auf die das Muster zutrifft, als Treffer identifiziert. Um die kürzestmögliche Zeichenkette zu finden, muss ausgeschlossen werden, dass die Zeichenkette ein weiteres „n“ enthält:

```
match("Apfel und Birne und Orange", /A[^n]*n/);
```

Ergebnis:

```
Apfel un
```

Die zu verarbeitende Zeichenkette kann auch in einer Systemvariablen ($0, $1 etc.) enthalten sein. Beispiel:

```
match($0, /A.*n/);
print NR, substr($0,RSTART,RLENGTH);
```

Dieser Code-Abschnitt gibt beim Durchlaufen einer ↗Eingabedatei sämtliche Zeichenketten, auf die das Suchmuster passt, aus und schreibt unmittelbar davor die Nummer der ↗Zeile (NR), in der das Muster gefunden wurde.

3.5.3.8 Aussagenlogik

Im Zusammenhang mit der Formulierung von Bedingungen, etwa für die Ausführung von Code-Blöcken, wird man mit einem aus Sicht der Alltagswelt ungewohnten Konzept konfrontiert: der Aussagenlogik. Wir beschränken uns an dieser Stelle auf die für die korpuslinguistische Praxis relevanten Konzepte, nämlich die Konjunktion, die Adjunktion sowie die Disjunktion. Als exemplarische Datenbasis liege folgende ↗Tabelle vor:

ID	Feld 1	Feld 2	Konjunktion	Adjunktion	Disjunktion
1	Das	ART		x	x
2	der	ART	x	x	
3	die	ART		x	x
4	der	ART	x	x	
5	der	ART	x	x	
6	der	ART	x	x	
7	die	ART		x	x
8	der	PRELS		x	x
9	die	ART		x	x
10	Der	ART		x	x
11	der	ART	x	x	

12	die	ART		x	x
13	die	ART		x	x
14	das	PDS			
15	die	ART		x	x
16	der	ART	x	x	
17	das	ART		x	x
18	die	PRELS			
19	der	ART	x	x	
20	das	ART		x	x
21	die	ART		x	x
22	der	ART	x	x	

- Konjunktion: Finde alle Datensätze, in denen Feld 1 den Wert „der“ und Feld 2 den Wert „ART“ besitzt. In AWK lautet der entsprechende logische Operator „&&“. Im Rahmen einer if-Anweisung wäre also zu formulieren:

```
if ($1=="der" && $2=="ART")
```

- Adjunktion (= „nichtausschließendes Oder“): Finde alle Datensätze, die entweder im Feld 1 „der“ oder im Feld 2 „ART“ aufweisen. In AWK-Syntax wird in diesem Fall der Operator „||“ verwendet:

```
if ($1=="der" || $2=="ART")
```

- Disjunktion (= „ausschließendes Oder“): Finde alle Datensätze, die entweder im Feld 1 „der“ oder im Feld 2 „ART“, aber nicht sowohl im Feld 1 „der“ als auch, gleichzeitig, im Feld 2 „ART“ aufweisen. Der Operator „!“ fungiert als Zeichen der Negation. In AWK-Syntax:

```
if ((a[1]=="der" || a[2]=="ART") && !(a[1] == "der" &&
a[2] == "ART"))
```

Oben abgebildete ↗Tabelle illustriert sehr schön, dass sich Konjunktion und Disjunktion zueinander komplementär verhalten, während die Adjunktion beide miteinander verbindet. Im Anhang (S. 209) findet sich ein AWK-Skript, das das hier gegebene Beispiel in ausführbarem Programmcode enthält.

Im Sinne der Aussagenlogik wäre bei der umgangssprachlich möglicherweise so formulierten Suche: „Finde alle ↗Tokens ‚der‘ und ‚die‘“,

das „und“ mit dem logischen Operator „||“ („oder“) und nicht „&&“ („und“) wiederzugeben.

4 Digitalisierung von Sprache und Text

4.1 Digitalisierung von gesprochener Sprache: Das Programm Praat

Das Programm Praat ist von Phonetikern für Phonetiker gemacht und besitzt einen beeindruckenden Funktionsumfang. Aus Sicht der Korpuslinguistik ist es vor allem deswegen interessant, weil damit Audio-Aufnahmen von Sprache transkribiert und für die weitere Verarbeitung mit Skripts und in ↗Datenbanken aufbereitet werden können.

Praat ist kostenlos und für alle gängigen ↗Betriebssysteme verfügbar. Es kann unter http://www.fon.hum.uva.nl/praat/ heruntergeladen werden. Praat wird ständig aktualisiert. Man sollte sich stets die neueste Version von der Webseite der Entwickler herunterladen.

Voraussetzung für die Arbeit mit Praat ist das Vorhandensein von geeigneten Audioaufnahmen. Beachten Sie unbedingt die allgemeinen Regeln hinsichtlich der systematischen Strukturierung des Korpusmaterials (s. oben). Wir empfehlen, sich an der im Asica-Korpus verwendeten Systematik zu orientieren (http://asica.gwi.uni-muenchen.de).

Die Audioaufnahmen sollten im wav-Format vorliegen. Zwar kann Praat mittlerweile auch mit mp3-Dateien arbeiten, die Zeitreferenzierung ist dann aber bis zu mehreren Millisekunden ungenau.

Das folgende Beispiel präsentiert die Transkription einer Audioaufnahme der Wenkersätze.

1. Starten des Programms

Das Programm wird durch Doppelklick gestartet. Praat öffnet automatisch zwei Fenster, nämlich „Praat Objects“ und „Praat Picture“. Letzteres wird nicht benötigt und kann gleich wieder geschlossen werden.

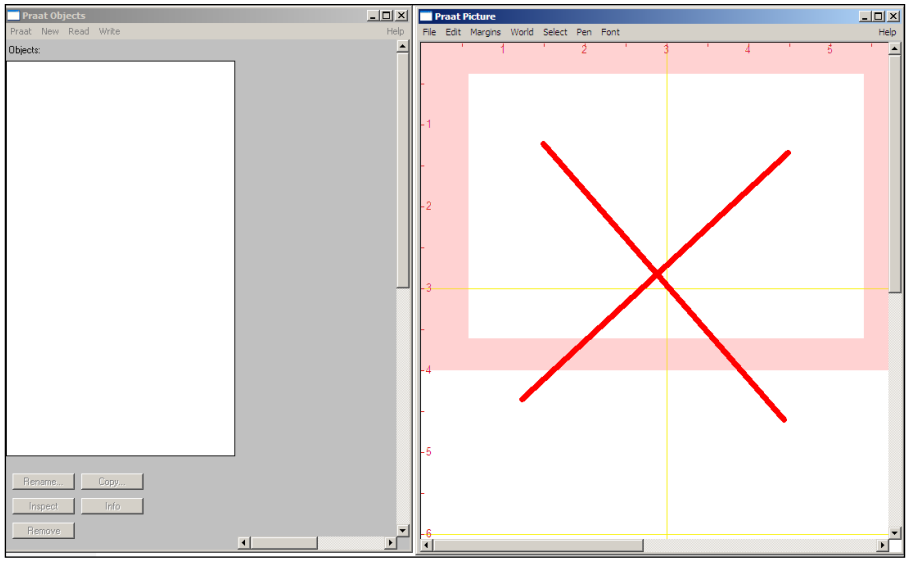


Abbildung 19: Praat nach dem Programmstart: Das Praat-Objects-Fenster links, rechts das Praat-Picture-Fenster

2. Öffnen der Audio-Datei aus Praat-Objects heraus

Gehen Sie auf „Read“ und dann auf „Open long sound file ...“

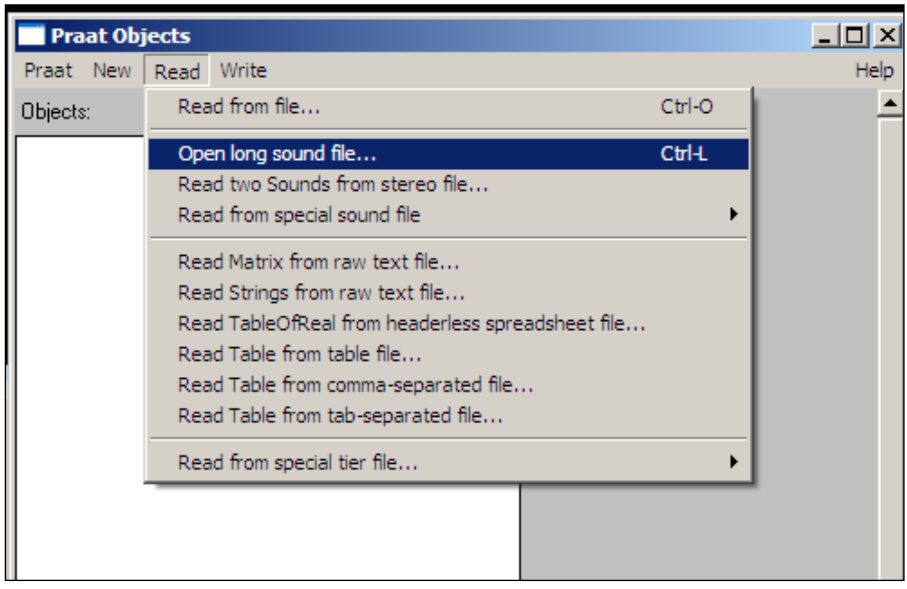


Abbildung 20: Öffnen einer Audiodatei mit Praat

3. Markieren Sie die geöffnete Audio-Datei in der Liste der „Objects“ und klicken Sie auf „Annotate“ und anschließend auf „To TextGrid...“

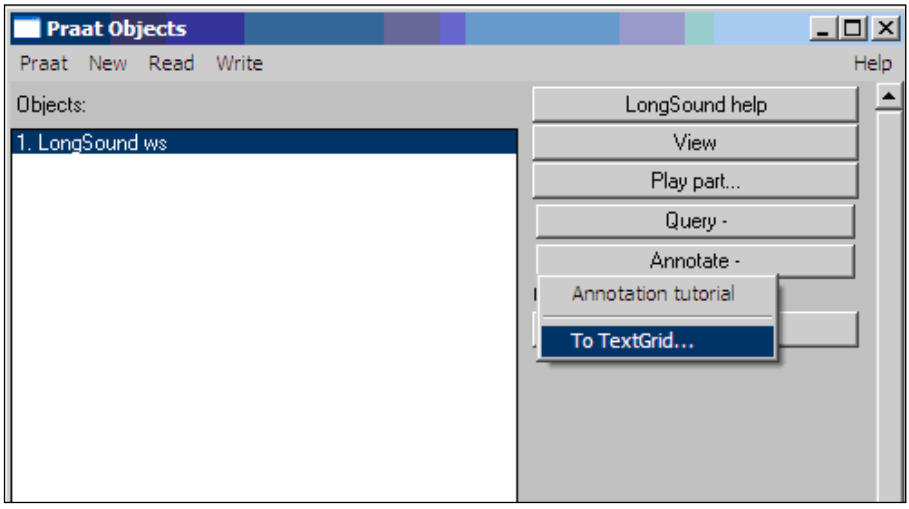


Abbildung 21: Erzeugung eines TextGrids in Praat

4. Praat fragt dann nach den Namen der sog. „Tiers“. Damit sind zunächst Ebenen gemeint, die für die separate Transkription der Äußerungen verschiedener Sprecher gedacht sind. Im Sinne der Korpuslinguistik hat es sich in vielen Fällen als günstig erwiesen, unterschiedliche grammatische Kategorien diesen Ebenen zuzuordnen. Entsprechend würde man „Satz“, „Wort“ und „Silbe“ als Tier-Namen wählen. Die Entscheidung ist natürlich vom jeweils verfolgten Forschungsinteresse abhängig.

In den meisten Fällen ist ein Tier, in das der Text satzweise transkribiert wird, ausreichend:

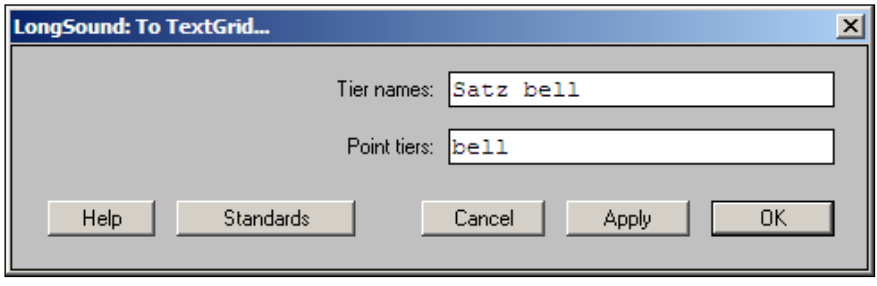


Abbildung 22: Dialogfeld zur Anlage und Benennung der „Tiers“ in einem TextGrid

Weitere Tiers können auch nachträglich noch hinzugefügt werden.

Sollte ein Tier durch zusätzlichen Eintrag seines Namens in die zweite Zeile als „Point tier“ gekennzeichnet werden (hier das Tier „bell“), so können in diesem Tier keine Zeitintervalle, sondern nur Zeit**punkte** definiert werden.

Nach dem Klick auf „OK“ wird der Objektliste ein neuer Eintrag hinzugefügt, nämlich „TextGrid xx“:

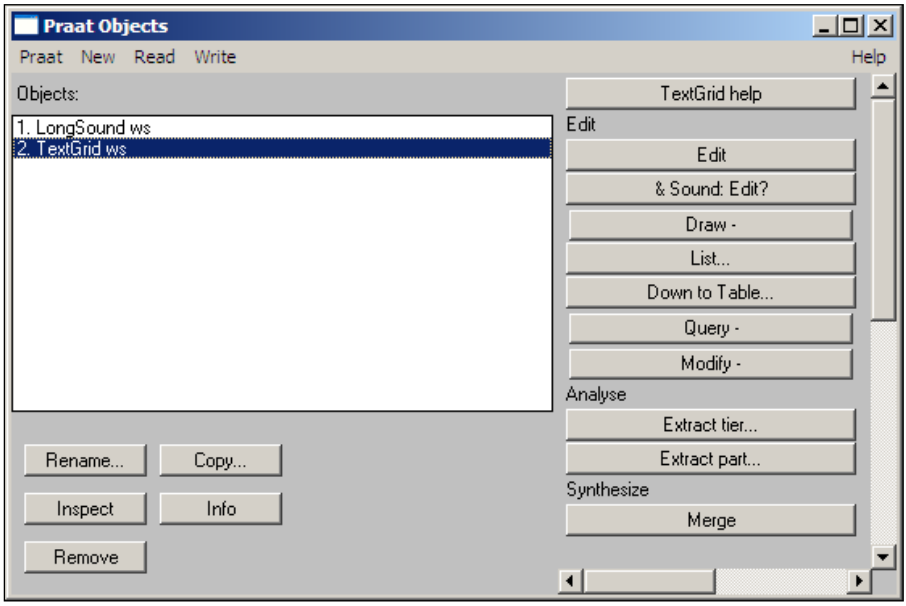


Abbildung 23: Das Praat-Objekt-Fenster mit den Einträgen für die Audiodatei und das zugehörige TextGrid

Nun müssen beide Listeneinträge gemeinsam markiert werden (dazu hält man beim Mausklick die Strg-/Ctrl-Taste gedrückt):

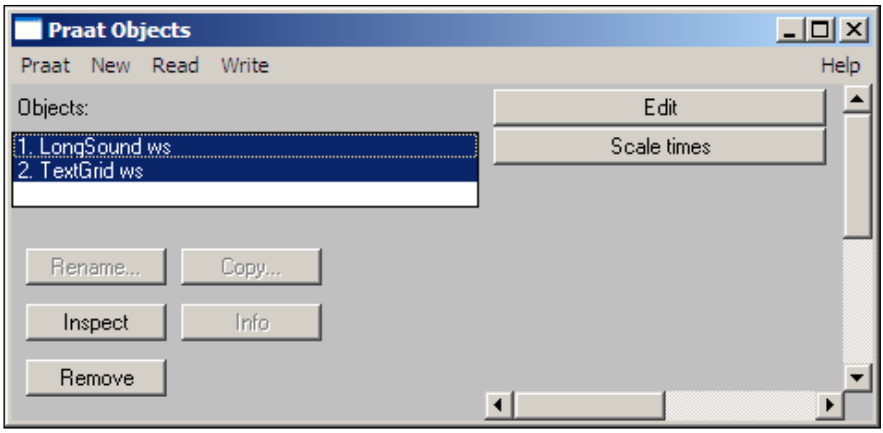


Abbildung 24: Gleichzeitige Auswahl von Audiodatei und TextGrid

Ein Klick auf „Edit“ öffnet folgende Ansicht:

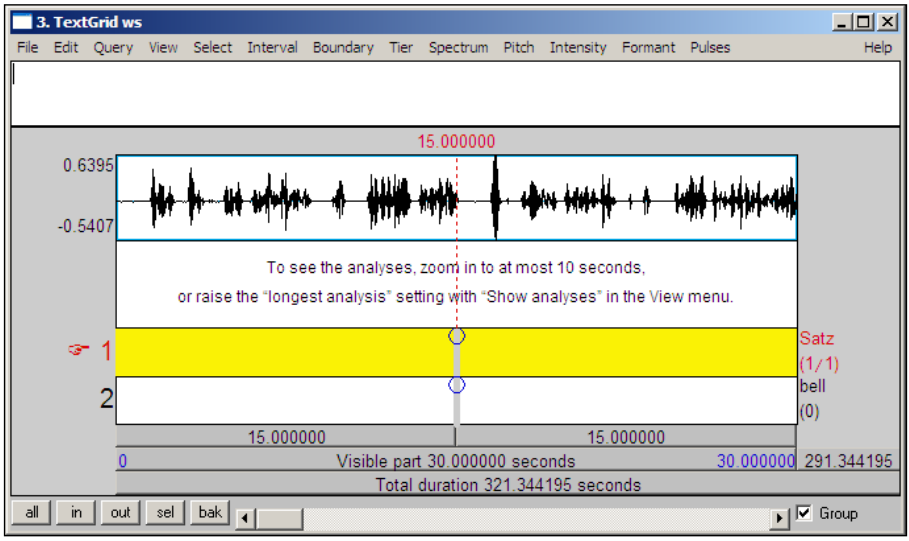


Abbildung 25: Praat-Fenster zur Transkription einer Audiodatei

Die Größe des Ausschnitts läßt sich mit den kleinen Knöpfen am linken unteren Fensterrand verändern.

Nun folgt ein wichtiger Schritt: die Segmentierung der Audiodatei in einzelne Intervalle. Je kleinteiliger die Segmentierung erfolgt, desto präziser kann später, etwa bei Verwendung einer ↗Datenbank, auf den jeweils zu einem Textabschnitt gehörigen Audioabschnitt zugegriffen werden. Eine kleinteiligere Segmentierung bedeutet jedoch auch einen

größeren Arbeitsaufwand. Auf jeden Fall sollte die Segmentierung so erfolgen, dass die Intervalle stets Inhalte gleicher Kategorie besitzen, will heißen:

Satz | Satz | Satz | ...

oder

Wort | Wort | Wort | ...

oder

Silbe | Silbe | Silbe | ...

NICHT:

Satz | Silbe | Wort | Satz | ...

Bei gesprochener Sprache ist die Definition von Sätzen meist sehr problematisch. In der Praxis hat sich die intuitive Abgrenzung von „Äußerungseinheiten“ durchaus bewährt. Auf alle Fälle sollten, sofern möglich (also bei nicht gleichzeitigem Sprechen), Passagen unterschiedlicher Sprecher in jeweils eigene Intervalle gesetzt werden.

Das Abspielen des Tons wird durch Drücken der Tabulator-Taste ausgelöst. Praat spielt nur Ausschnitte von maximal 60 Sekunden Länge ab. Bei längeren Ausschnitten erfolgt eine entsprechende Fehlermeldung:

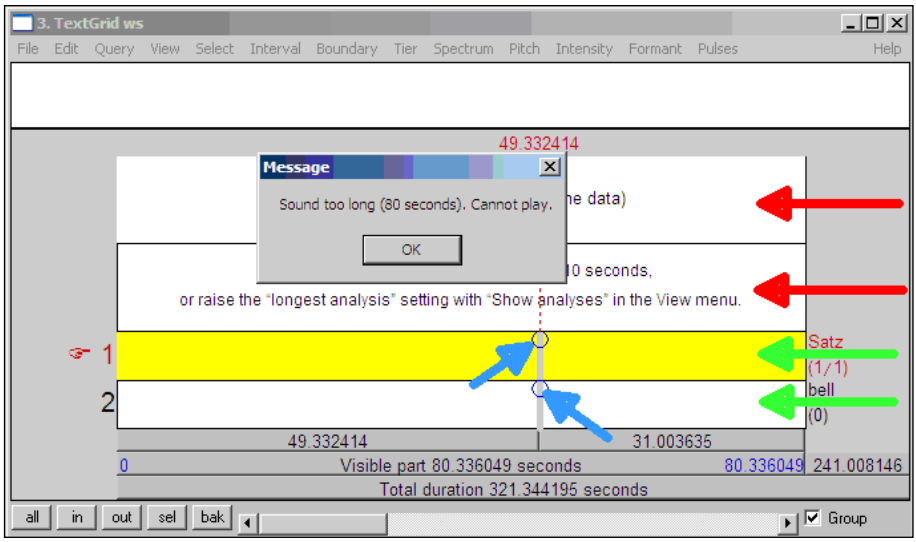


Abbildung 26: Die Anzeige- und Funktionselemente im Praat-Transkriptionsfenster; Definition von Intervallen

Die Definition eines Ausschnitts erfolgt durch das Ziehen der Maus bei gedrückter linker Maustaste, wobei die Maus im Bereich der hier mit roten Pfeilen markierten Bereiche geführt werden muss.

Um eine Intervallgrenze zu setzen, muss man mit der linken Maus wiederum in einen der beiden mit Pfeilen markierten Bereiche klicken. Es erscheint dann (außer einer horizontalen, für uns irrelevanten) eine gestrichelte vertikale rote Linie, die jeweils an ihrem Schnittpunkt mit den oberen Begrenzungslinien der Tiers (grüne Pfeile) einen Kreis aufweist (blaue Pfeile). Ein Klick in diesen Kreis erzeugt eine Intervallgrenze im entsprechenden Tier.

Die Intervallgrenzen werden durch vertikale Linien im entsprechenden Tier gekennzeichnet:

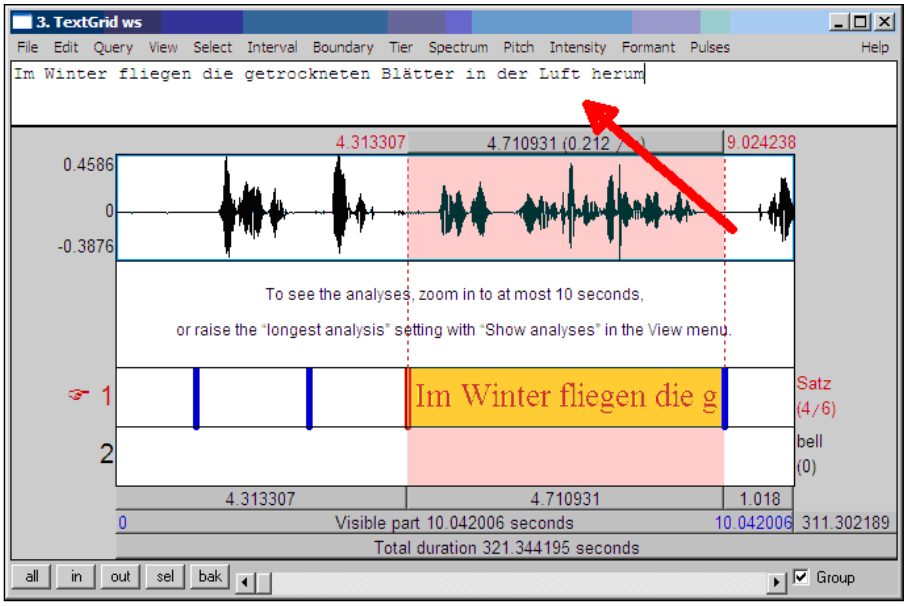


Abbildung 27: Transkription eines Audio-Intervalls

Die Auswahl eines Intervalls erfolgt durch Mausklick in den Bereich zwischen zwei vertikale Linien (= Intervallgrenzen). Anschließend erscheint das Intervall rot hinterlegt, und es ist möglich, in den weißen Bereich (roter Pfeil) am oberen Rand des Fensters den Transkriptionstext einzugeben. Schon bei der Texteingabe erscheint der Text zusätzlich im entsprechenden Bereich des Tiers (in roter Schrift).

Praat erlaubt es nicht, phonetische Sonderzeichen zu verwenden. Erforderlichenfalls müssen diese durch eine Abfolge von ↗ASCII-Zeichen eingegeben werden. Die zu verwendenden Zeichentabellen können durch Klicken auf das Hilfe-Menü abgerufen werden:

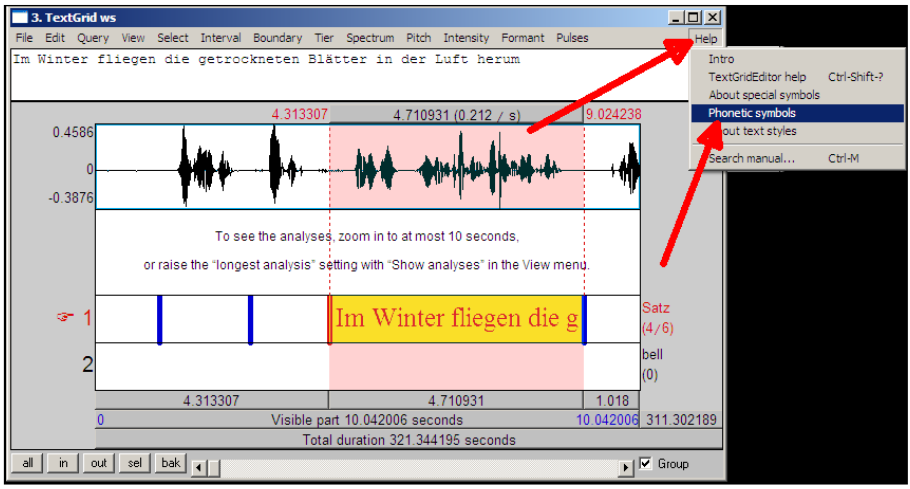


Abbildung 28: Aufruf der Praat-Schemata mit Zeichenkodierungskonventionen

Hier ein Ausschnitt aus der Konsonanten-Schemata:

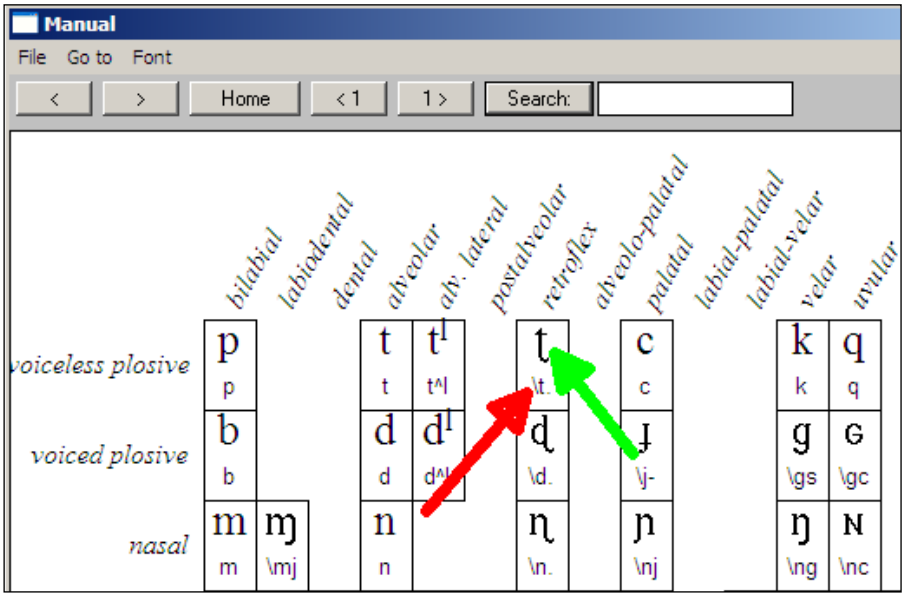


Abbildung 29: Die Praat-Konventionen zur Kodierung von Konsonanten

Der „retroflexe stimmlose Plosivlaut“, dessen phonetisches Symbol hier mit dem grünen Pfeil markiert ist, muss in Praat mit der Zeichenfolge „\t.“ wiedergegeben werden.

Nach Abschluß der Transkriptionsarbeit kann der Text in eine sog. „TextGrid-Datei“ exportiert werden. Es empfiehlt sich, den Text zuvor gemäß ↗Unicode zu kodieren. Dieser Schritt erfolgt über das Menü „Edit“ ⇒ „Convert entire TextGrid to Unicode“:

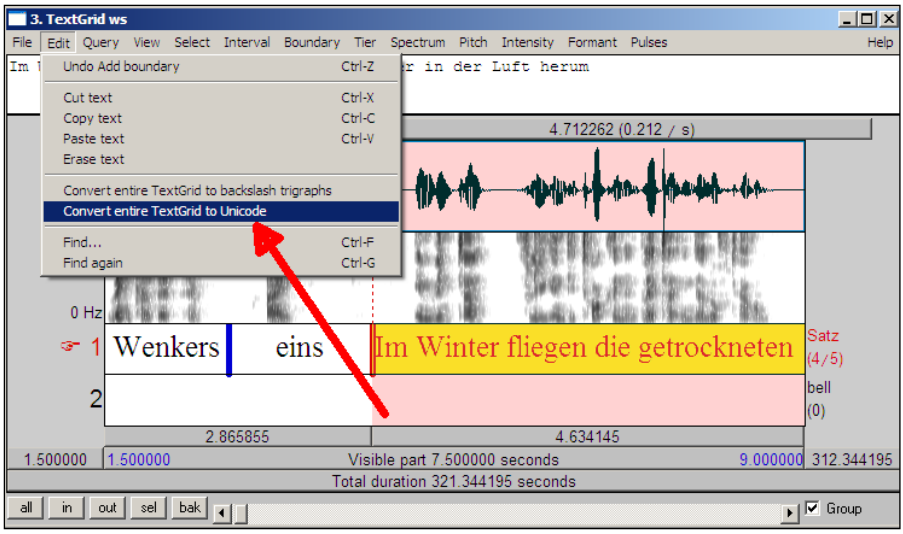


Abbildung 30: Umwandlung der Praat-spezifischen Zeichenkodierung in ↗Unicode

Anschließend wird die Transkription über das Menü „File“ und „Write TextGrid to Text file“ in einer Textdatei geschrieben:

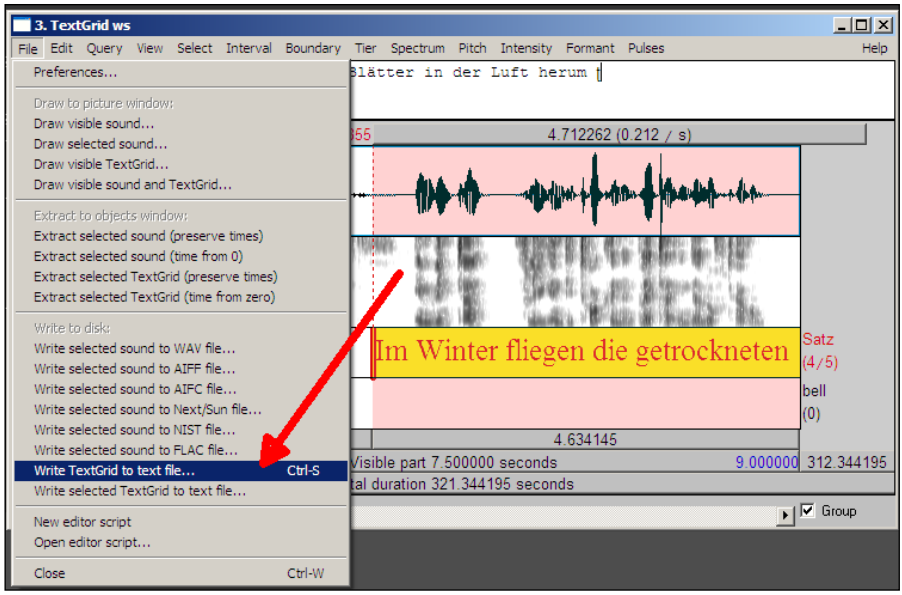


Abbildung 31: Export eines TextGrids in eine Textdatei

Die dabei entstandene Datei kann anschließend z.B. mit einem AWK-Skript in ↗Tabellenform gebracht und dann in eine ↗Datenbank importiert werden:

```
File type = "ooTextFile"
Object class = "TextGrid"

xmin = 0
xmax = 321.3441950113379
tiers? <exists>
size = 2
item []:
    item [1]:
        class = "IntervalTier"
        name = "Satz"
        xmin = 0
        xmax = 321.3441950113379
        intervals: size = 5
        intervals [1]:
            xmin = 0
            xmax = 1.0630635245901638
            text = ""
        intervals [2]:
            xmin = 1.0630635245901638
            xmax = 2.8301618852459014
            text = "Wenkersätze"
        intervals [3]:
            xmin = 2.8301618852459014
            xmax = 4.365854508196721
            text = "eins"
        intervals [4]:
            xmin = 4.365854508196721
            xmax = 9.078116803278688
            text = "Im Winter fliegen die getrockneten Blätter in der Luft herum"
        intervals [5]:
            xmin = 9.078116803278688
            xmax = 321.3441950113379
            text = ""
    item [2]:
        class = "TextTier"
        name = "bell"
        xmin = 0
        xmax = 321.3441950113379
        points: size = 0
```

Abbildung 32: Eine Praat-TextGrid-Datei, geöffnet im Editor VIM

ACHTUNG: Die von Praat erzeugte TextGrid-Datei in ↗Unicode-Kodierung ist möglicherweise (mindestens bis Version 5.2.25 standardmäßig) ↗UTF-16-kodiert. Es wird dringend empfohlen, die TextGrid-Dateien vor der Weiterverarbeitung nach ↗UTF-8 zu konvertieren.

4.2 Digitalisierung von gedrucktem Text: Das Programm FineReader

4.2.1 Einführung

FineReader (FR) gehört zur Gruppe der OCR-Programme: Optical Character Recognition – Optische Zeichenerkennnung. Es existiert nur in einer Windows-Version und ist ein kommerzielles Produkt der Firma AB-

BYY. Lizenzen für Forschung und Lehre werden relativ kostengünstig angeboten.

OCR-Programme verwandeln Graphik in computerlesbaren Text. Während in einer Graphik-Datei ein Buchstabe durch eine Vielzahl einzelner Bildpunkte repräsentiert wird (die Anzahl der Bildpunkte hängt von der Größe des Buchstabens ab, für jeden Bildpunkt wird dessen Position und Farbe notiert), steht in einer Textdatei für einen Buchstaben lediglich eine einzige Zahl. Ein OCR-Programm muss dazu in der Lage sein, die Zusammengehörigkeit von Bildpunkten zu erkennen und den richtigen Zahlenwert gemäß einer standardisierten ↗Codepage (↗ASCII, ↗Unicode) zu ermitteln.

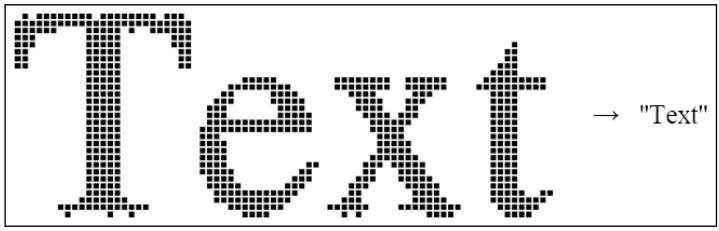


Abbildung 33: Verwandlung von Bildpunkten in elektronischen Text[39]

OCR spielt in der Korpuslinguistik vor allem dann eine Rolle, wenn das Korpusmaterial in auf Papier gedruckter Form vorliegt. Der erste Schritt besteht dann in der Digitalisierung des Textes durch Photographieren oder Scannen (scannen ist photographieren vorzuziehen, da bei Letzterem die horizontale und gerade Ausrichtung der Textzeilen häufig nicht gegeben ist). Es ist ganz wesentlich, diesen Arbeitsschritt mit großer Sorgfalt und systematisch durchzuführen!

- Trennen Sie die Bilderfassung mittels Scanner oder Photoapparat vom Prozeß der Texterkennung mit OCR-Software.
- Achten Sie auf eine systematische und aussagekräftige Benennung der Bilddateien.
- Sofern Sie kein einschlägiges Interesse verfolgen, wählen Sie hinsichtlich der Farberfassung „Graustufen“ und verzichten Sie auf Farben (Farberfassung erzeugt unnötig große Dateien, da

[39] Quelle: http://www.lrz.de/services/peripherie/scanner/scantips/. Die dort zusammengestellten Tipps sind ausdrücklich zur Lektüre empfohlen!

pro Bildpunkt nicht ein, sondern drei ↗Byte [= 24 ↗Bit] benötigt werden)

- Als Auflösung wählen Sie 300 dpi, bei kleiner Schrift u.U. auch 400 dpi
- Als Dateiformat wählen Sie TIF oder JPEG (bitte mit OCR testen! Siehe unten)
- Scannen Sie zunächst eine „repräsentative“, typische Seite des Textes und führen Sie einen Test durch, ob die OCR-Prozedur ein brauchbares Ergebnis liefert.

Nachdem Sie den kompletten Text erfaßt haben, beginnen Sie mit der OCR-Prozedur. Es ist durchaus eine Option, die Texterkennung auf einem Fremdrechner durchzuführen, auf dem eine lizensierte Version des OCR-Programmes installiert ist. Man spart sich damit die Anschaffung eines normalerweise nicht billigen Programmes.

4.2.2 OCR mit FineReader

Die hier beschriebene Prozedur bezieht sich auf FineReader (FR) 11.0 Professional Edition für Windows (unter Windows 7).

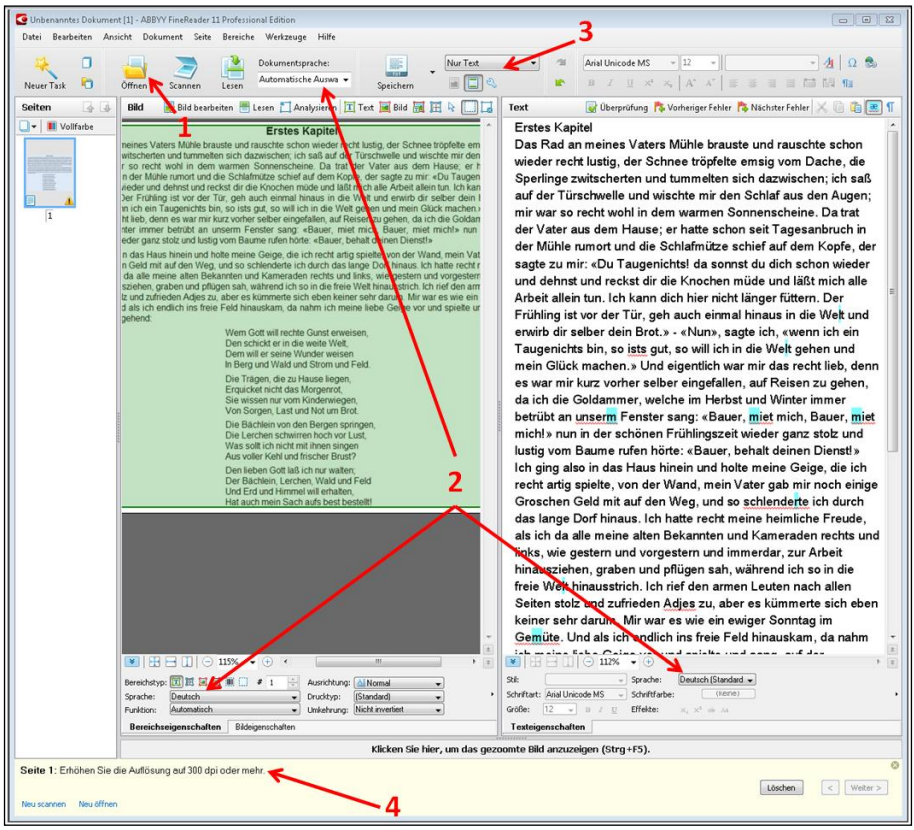


Abbildung 34: Die Standard-Oberfläche von FR 11 Professional. In der linken Bildhälfte die Bilddatei, in der rechten der bereits interpretierte, elektronische Text

Starten Sie das Programm und gehen Sie dann wie folgt vor:

1. Öffnen Sie die Bilddatei(en) (Pfeil 1 im Screenshot)

2. FR beginnt sofort mit der Texterkennung. Nach Abschluß dieses Vorgangs meldet FR in einem kleinen Fenster bzw. am unteren Fensterrand Fehler oder Warnungen (Pfeil 4). Nehmen Sie diese Hinweise bereits bei der Bearbeitung der ersten Probeseite zur Kenntnis, da die Probleme auf eine evtl. mindere Qualität der Bilddateien zurückzufüh-

ren sein können. In diesem Fall muss die Vorlage neu gescannt/photographiert werden!

Je einfacher das Layout und je standardnäher die Sprache, desto bessere Ergebnisse liefert FR. Zwischen sehr gut „lesbaren" Texten und nahezu „unlesbaren" gibt es eine stufenlose Skala mit Ergebnissen unterschiedlicher Qualität. Grundsätzlich gilt, dass nahezu jeder von FR erzeugte elektronische Text einer mehr oder minder umfangreichen Nachbearbeitung bedarf. Unter Umständen ist abzuwägen, ob eine manuelle Texteingabe („abtippen") nicht die effizientere Methode wäre.

3. Abspeichern des erkannten Textes (Pfeil 3). Für die Weiterverarbeitung mit Unix-Tools und schließlich den Import in eine ↗Datenbank muss als Format „Text-Dokument" und „Nur Text" ausgewählt werden (Pfeil 3)! Denken Sie beim Abspeichern wiederum an eine aussagekräftige und systematische Benennung der Dateien!

4.2.3 Tipps und Tricks

Um die Erkennungsgenauigkeit zu erhöhen, hilft – abgesehen von der Optimierung der Bildvorlage-Dateien – die Einstellung der richtigen Erkennungssprache (Pfeile 2 und 4). FR arbeitet unter anderem mit eingebauten Lexika. Es ist wichtig, dass FR auf das richtige Wörterbuch zurückgreift, also bei einem deutschen Text auf das deutsche Wörterbuch und nicht z.B. auf das englische. FR trifft die entsprechende Wahl automatisch und recht zuverlässig. Manchmal „irrt" sich das Programm aber auch, und dann muss die Erkennungssprache manuell eingestellt werden. FR erlaubt es auch, in verschiedenen Sprachen (z.B. absatz- bzw. abschnittsweise gemischt) zu erkennen.

Eine weitere Möglichkeit, die Erkennungsgenauigkeit zu erhöhen, ist das „Trainieren" von FR. Darunter versteht man den Vorgang, von FR erkannte Bildmuster bestimmten Schriftzeichen zuzuweisen. Diese Prozedur wird gestartet über das Menü „Werkzeuge" ⇒ Optionen ⇒ 2. Lesen ⇒ Test ⇒ Benutzermuster testen:

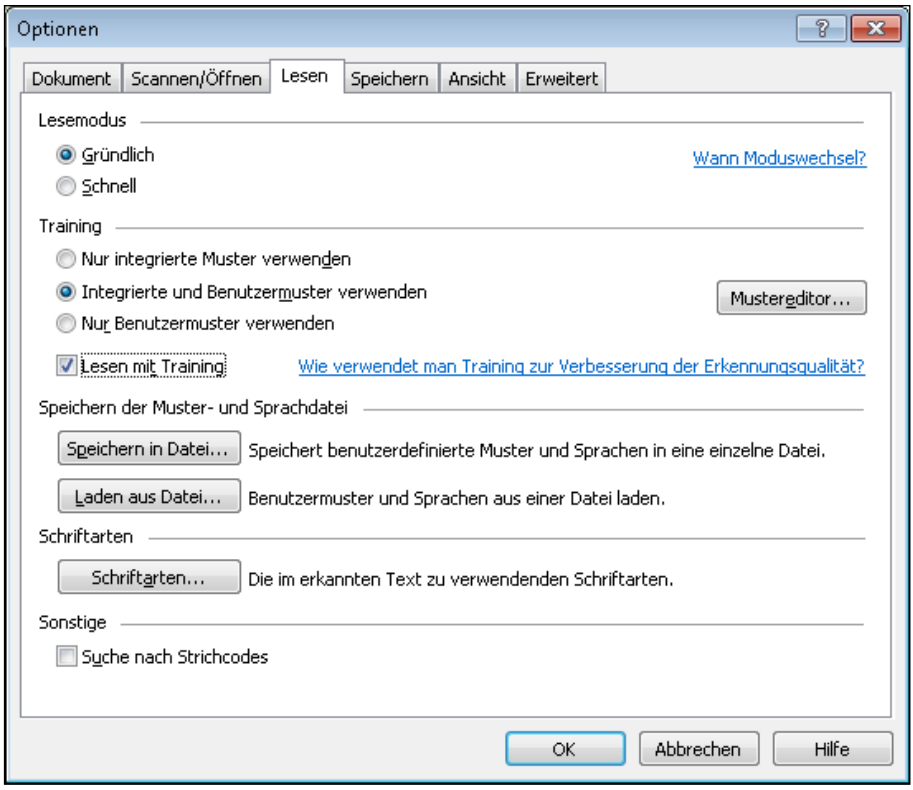


Abbildung 35: Start des Trainings der Mustererkennung zur Verbesserung der Leseergebnisse

Wenn man nun das Bild neu lesen läßt, fragt FR bei einzelnen von ihm als zusammengehörig erkannten Mustern, um welches Zeichen es sich handelt. FR unterbreitet jeweils einen Vorschlag, den man ggf. ändern kann. Ein Klick auf den Schalter „Training“ bewirkt die Abspeicherung der gewählten Zuordnung. Der von FR gewählte Muster-Ausschnitt kann durch Veränderung des grünen Rahmens erweitert oder verkleinert werden. Auf diese Weise wird das Musterrepertoire von FR erweitert bzw. gepflegt.

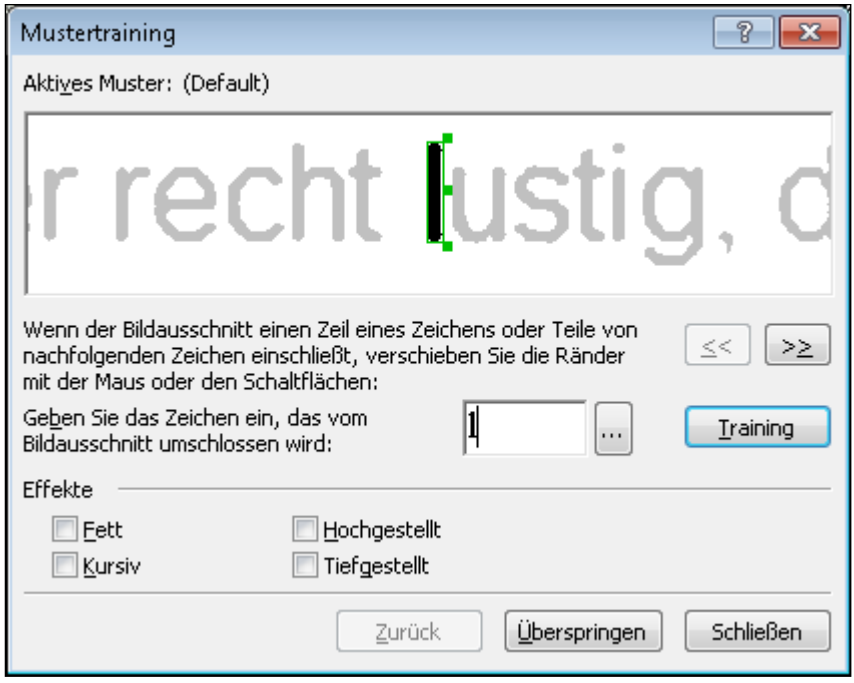


Abbildung 36: Trainieren der Zeichenerkennung

Dieses Training führt man so lange durch, bis sich das Leseergebnis spürbar verbessert hat. Anschließend wählt man im Dialogfeld Optionen den Punkt „Benutzermuster verwenden“.

Es kommt vor, dass man beim Training Fehler macht. Diese lassen sich im Mustereditor (Aufruf über Werkzeuge ⇒ Mustereditor... (gewünschtes Listenelement markieren) ⇒ Bearbeiten... ⇒ Details) wieder löschen bzw. verbessern (⇒ Eigenschaften):

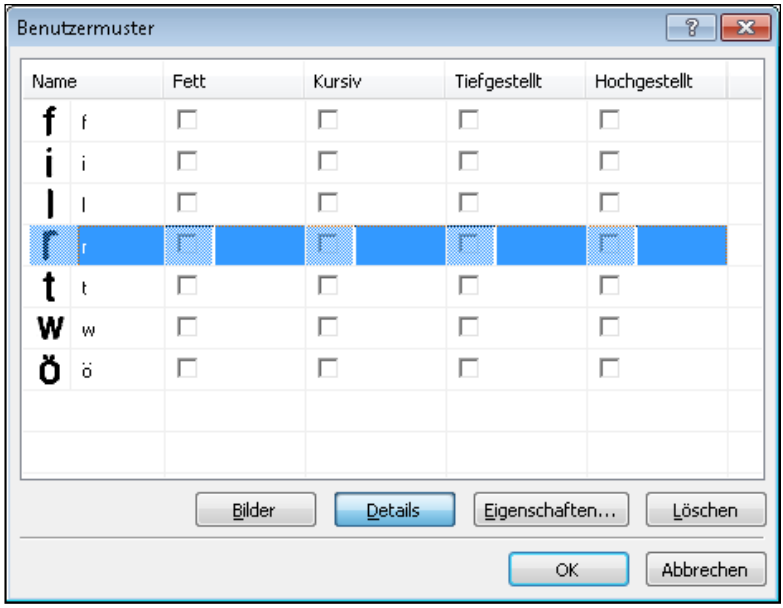


Abbildung 37: Nachbearbeitung trainierter Muster in FR

FR speichert diese Muster-Zeichen-Zuordnungen in eigenen Dateien ab, die man, speziell, wenn man viel Mühe auf ein Training verwendet hat, sichern sollte. Wo sich die jeweilige Musterdatei befindet und wie sie heißt, steht im Dialogfeld „Mustereditor“:

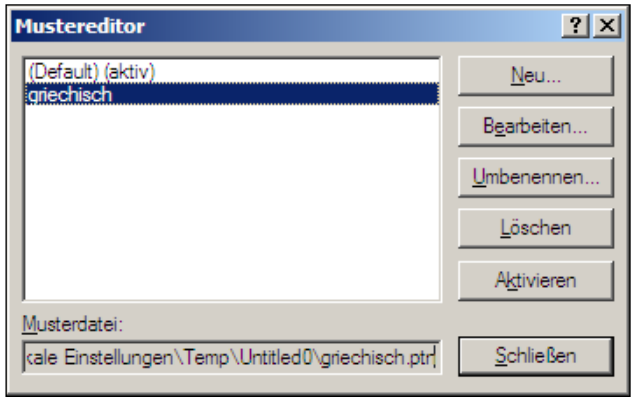


Abbildung 38: Abspeichern einer individuell erzeugten Musterdatei

Ein spezielles Problem stellen Texte in nicht-lateinischen Alphabeten dar. Zwar „liest“ FR ohne weiteres solche Texte (z.B. Russisch), sobald man aber ein Mustertraining durchführen möchte, stößt man auf das

Problem, dass man die benötigten Zeichen nicht über die Tastatur eingeben kann (z.B. ein griechisches Alpha mit Iota subscriptum). Diesem Problem kommt man mit einem Trick bei: Man führt mit der betreffenden Bilddatei ein Training durch und gibt jeweils den ↗Unicode-Zahlenwert der Zeichen ein, und zwar in der vom ↗HTML-Standard vorgeschriebenen Form (HTML-Entity):

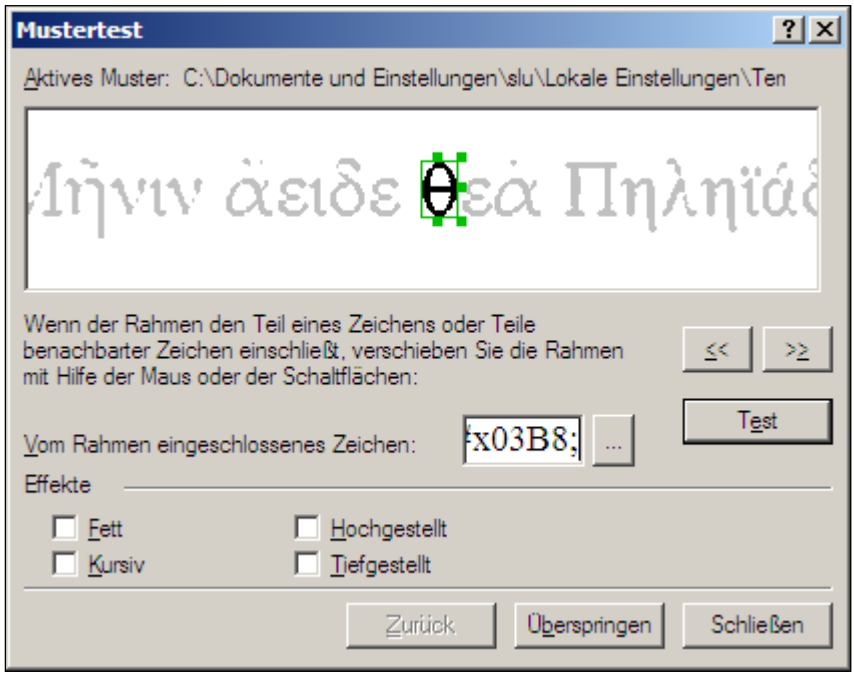


Abbildung 39: Verwendung von ↗Unicode-Werten in Verbindung mit ↗HTML-Entities bei Training und Erzeugung einer Musterdatei

Das griechische Theta ist der ↗Unicode-↗Codepage zufolge dem Zahlenwert x03B8 zugeordnet. Der ↗HTML-Standard verlangt folgende Schreibweise: θ – Genau diese Zeichenfolge wird im Muster-Fenster in das Feld „Vom Rahmen eingeschlossenes Zeichen:“ eingetragen und mit Klick auf „Test“ abgespeichert. FR interpretiert die Zeichenfolge als Ligatur. Die entsprechende Nachfrage muss man mit „Ja“ bestätigen:

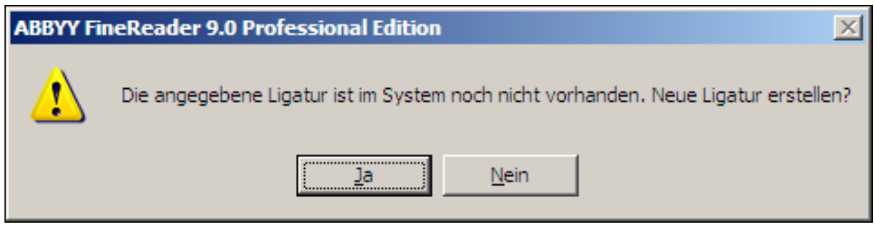


Abbildung 40: Definition einer HTML-Entity als (Pseudo-)„Ligatur“

Nach Abschluß des Trainings sieht die Musterzuordnung folgendermaßen aus:

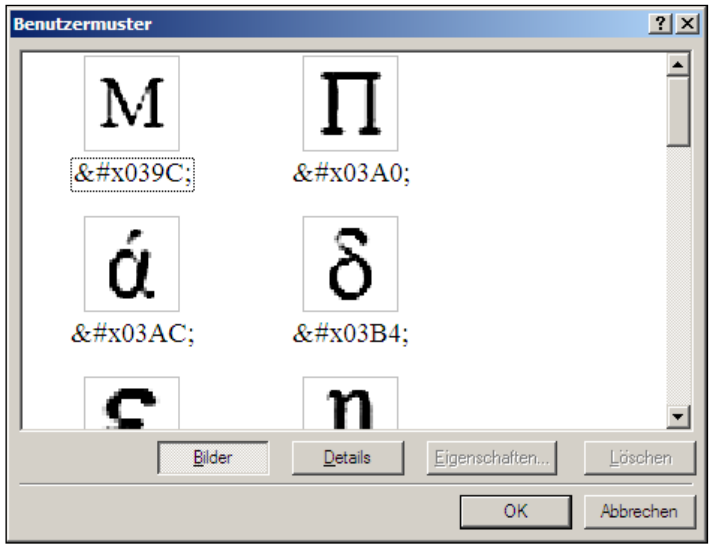


Abbildung 41: FR-Benutzermuster mit Pseudo-Ligaturen zur Kodierung von Sonderzeichen

Wenn man nun den griechischen Text von FR unter Verwendung dieser Musterdatei lesen läßt, erzeugt FR einen ↗HTML-konformen Text (rechts):

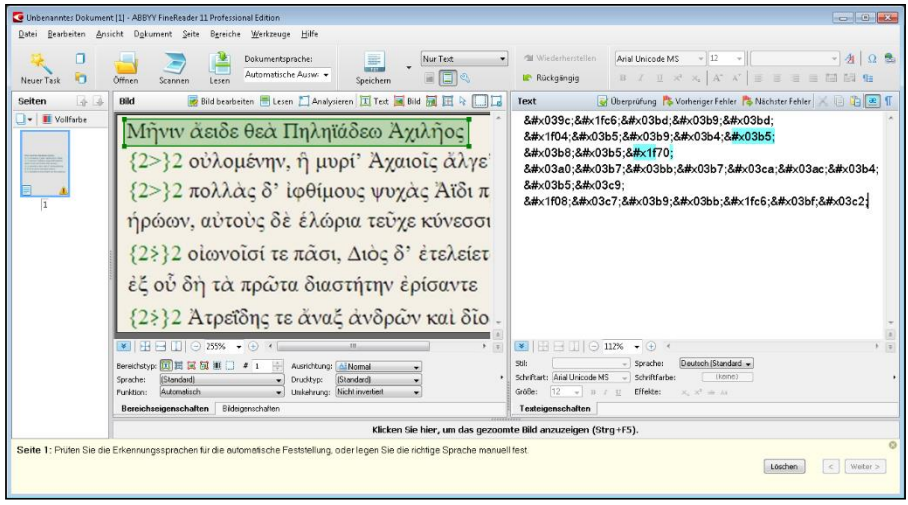


Abbildung 42: „Übersetzung“ einer griechischen Textvorlage in Bildgestalt in elektronischen Text in ↗HTML-konformer Kodierung (↗HTML-Entities mit ↗Unicode-Werten)

Diesen Text speichert man dann im Format „Nur Text“ ab. Das Ergebnis sieht dann so aus:

```
&#x039c;&#x1fc6;&#x03bd;&#x03b9;&#x03bd;
&#x1f04;&#x03b5;&#x03b9;&#x03b4;&#x03b5;
&#x03b8;&#x03b5;&#x1f70;
&#x03a0;&#x03b7;&#x03bb;&#x03b7;&#x03ca;&#x03ac;&#x03b4;
&#x03b5;&#x03c9;
&#x1f08;&#x03c7;&#x03b9;&#x03bb;&#x1fc6;&#x03bf;&#x03c2;
```

Wenn man nun diesen Text mit dem Internet-Browser Firefox öffnet, verwandelt Firefox die ↗HTML-Zeichen (korrekt: „↗HTML-Entities“) in griechische Buchstaben:

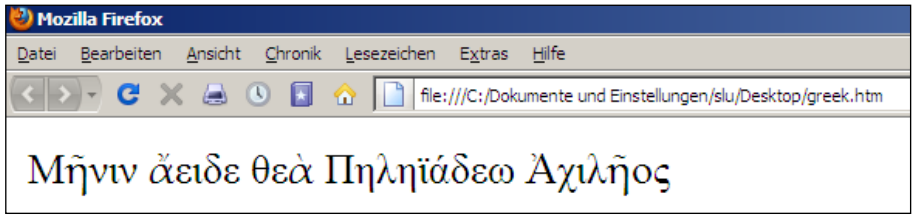


Abbildung 43: „Rückübersetzung“ der ↗HTML-Entities in griechischen Text mit Hilfe des Browsers Firefox

Über die copy-paste-Funktion läßt sich dieser Text dann z.B. in ein Word-Dokument einfügen und die Datei anschließend im ↗UTF-8-Format abspeichern.

Das Verfahren hat den Vorteil, dass sämtliche Zeichen eindeutig kodiert sind. Es entsteht eine solide Basis für die weitere Verarbeitung, z.B. in einer ↗Datenbank. Außerdem vermeidet man Probleme, die sich daraus ergeben, dass nicht-lateinische Zeichen nicht über die Standardtastatur eingegeben werden können (etwa beim Muster-Training in FR).

4.2.4 Nicht-lateinische Alphabete mit FR-eigenen Zeichensätzen

Nicht-lateinische Alphabete können auch mit FR-eigenen Zeichensätzen dargestellt werden.

Verfahren:

1. Öffnen des Spracheditors:

Extras ⇒ Spracheditor ...

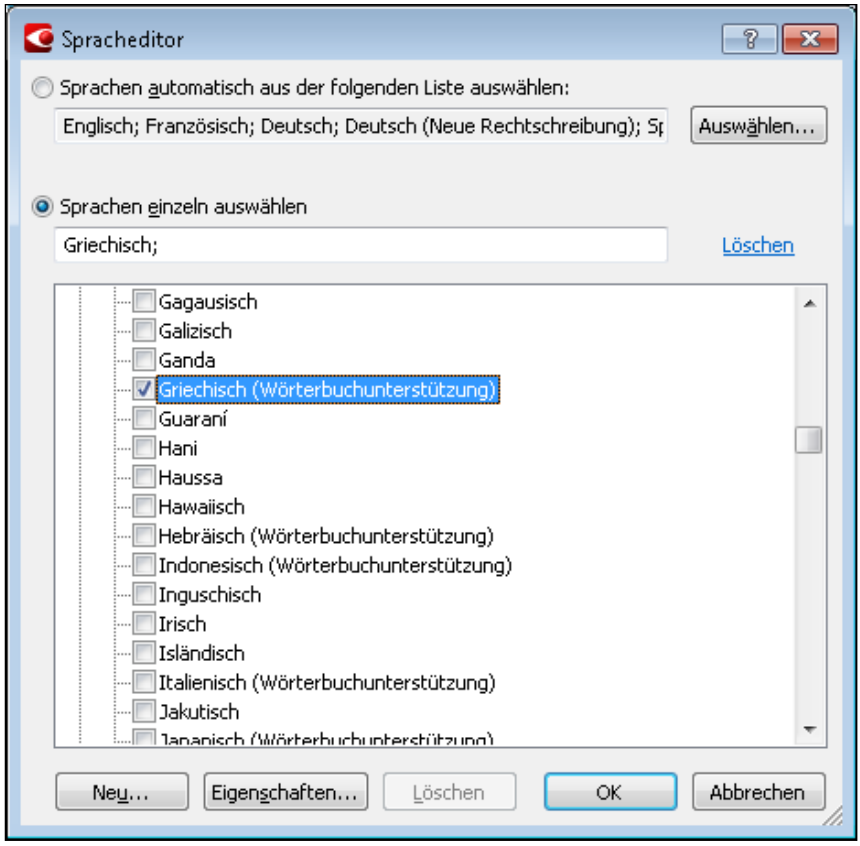


Abbildung 44: Auswahl von Erkennungssprachen in FR

2. Klick auf „Neu...“:

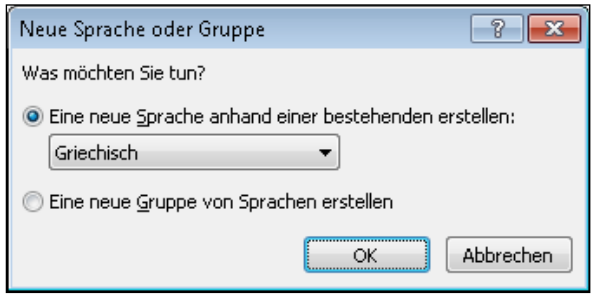


Abbildung 45: Definition einer neuen Erkennungssprache in FR

Anschließend öffnet sich das Dialogfeld „Spracheigenschaften“. Dort muss ein Name der Sprache eingegeben werden. Entscheidend ist nun,

dass an dieser Stelle das von der Sprache verwendete Alphabet angepaßt werden kann. Dazu muss man im Dialogfeld Spracheigenschaften auf die drei Punkte rechts neben Alphabet klicken:

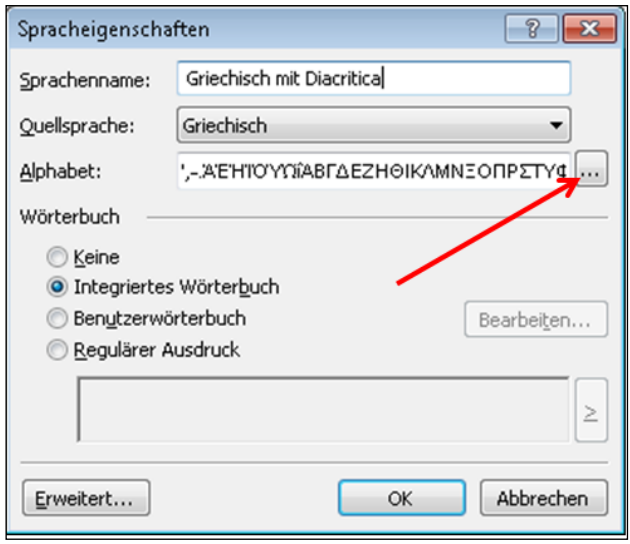


Abbildung 46: Benennung einer neuen Erkennungssprache sowie Anpassung des verwendeten Alphabets

Nun öffnet sich ein Dialogfeld, in dem die zusätzlich benötigten Zeichen des Alphabets ausgewählt werden können:

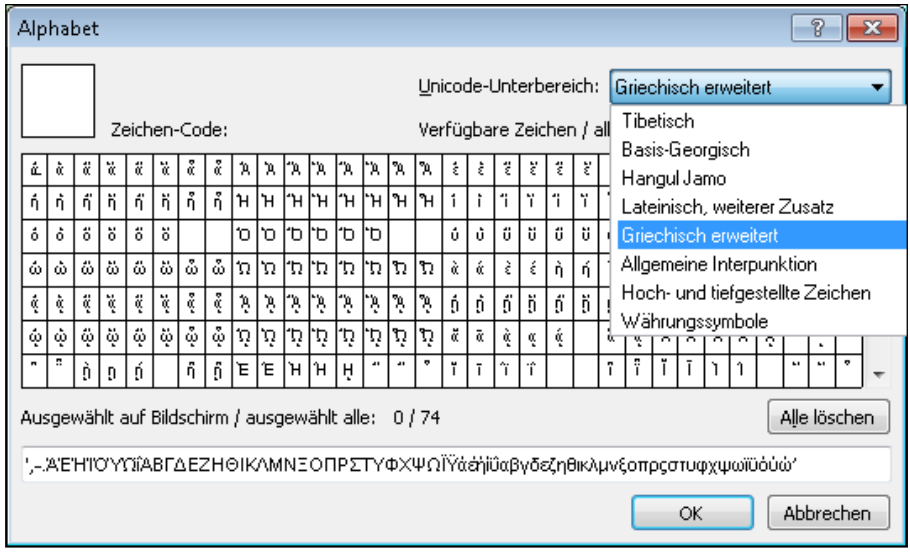


Abbildung 47: Zeichenauswahl für das erkennungssprachenspezifische Alphabet

Nun müssen die benötigten Zeichen ausgewählt werden. Man kann sie einzeln anklicken oder auch mit gedrückter Maustaste ganze Bereiche überstreichen und auf diese Weise markieren:

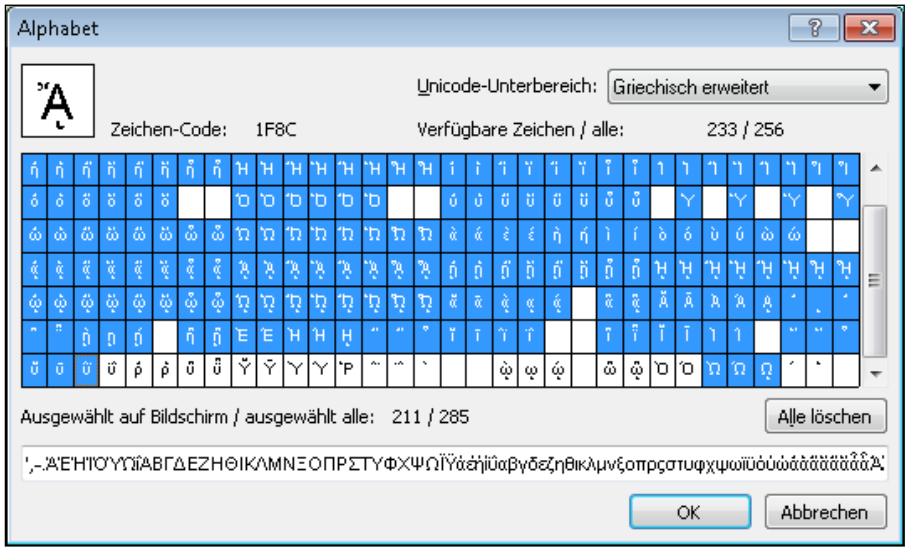


Abbildung 48: Für das Alphabet ausgewählte Zeichen

Anschließend müssen alle Dialogfelder mit „ok“ geschlossen werden.

Vor dem Start des Lesevorgangs sollte die Dokumentsprache auf die neu erstellte Sprache umgestellt werden.

Sollte das Leseergebnis unbefriedigend sein, muss noch ein passendes Zeichenmuster-Set angelegt werden:

Öffnen Sie den Mustereditor: ⇒ Werkzeuge ⇒ Mustereditor... ⇒ Neu...

Benennen Sie das Muster, aktivieren Sie es und wählen Sie in den Optionen „Benutzermuster testen“ aus. Anschließend starten Sie den Lesevorgang und beginnen mit dem Training:

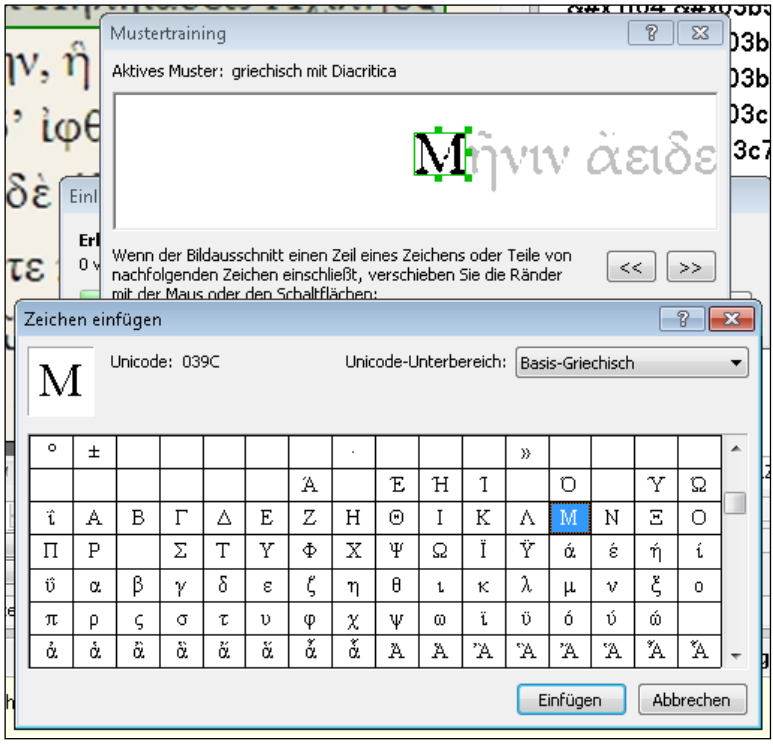


Abbildung 49: Verwendung der neu erstellten Erkennungssprache während des Mustertrainings

5 Tabellen und das relationale Datenmodell

5.1 Excel

Das Programm Excel, entwickelt von der Firma Microsoft und Bestandteil des Programmpakets „Office“, gehört zur Familie der sog. Tabellenkalkulationsprogramme. Es ist vor allem für Anwendungen im Bereich der Buchhaltung konzipiert, einige seiner umfangreichen Funktionen können aber auch für die Korpuslinguistik von Nutzen sein.

Wir beschränken uns auf die Beschreibung der Excel-Version 2007 für Windows.[40] Die Funktionsweise anderer Excel-Versionen, seien es ältere oder jüngere bzw. für andere ↗Betriebssysteme wie MacOS X, ist weitgehend analog und sollte auf Basis der vorliegenden Erläuterungen problemlos abgeleitet werden können.

Grundvoraussetzung für den nutzbringenden Einsatz dieses Programms ist, dass das zu analysierende Korpusmaterial bereits in einer geeigneten ↗Tabellenstruktur vorliegt. Sofern das Material nicht ohnehin in Form einer oder mehrerer ↗Tabellen organisiert ist, muss eine entsprechende Struktur unter Anwendung oben beschriebener Prozeduren erzeugt werden.

Die Anforderungen an die ↗Tabellenstruktur des Materials sind im Grunde identisch mit den Anforderungen, die auch bei der Verwendung von ↗Datenbanken zu beachten sind. Die wichtigste Grundregel dabei ist, dass die Spalten einer ↗Tabelle in jeder ↗Zeile stets Inhalte gleicher Art aufweisen müssen. Was im Einzelnen unter „gleicher Art“ zu verstehen ist, hängt stark von den jeweils verfolgten Zielen der Datenanalyse ab. Im folgenden Beispiel soll es um die morphosyntaktische Analyse gehen. Wir bedienen uns des Anfangs von Eichendorffs „Aus dem Leben eines Taugenichts“:

```
Das Rad an meines Vaters Mühle brauste und rauschte
schon wieder recht lustig, der Schnee tröpfelte emsig
vom Dache, die Sperlinge zwitscherten und tummelten sich
dazwischen;
```

[40] Inhaltlich sind die beiden Kapitel weitestgehend kongruent, so dass die Lektüre eines der beiden ausreichen sollte.

Eine für die Analyse mit Excel geeignete ↗Tabellenstruktur könnte in diesem Fall wie folgt aussehen:

Satz	Position	↗Token
1	1	Das
1	2	Rad
1	3	an
1	4	meines
1	5	Vaters
1	6	Mühle
1	7	brauste
1	8	und
1	9	rauschte
1	10	schon
1	11	wieder
1	12	recht
1	13	lustig
1	14	,
1	15	der
1	16	Schnee
1	17	tröpfelte
1	18	emsig
1	19	vom
1	20	Dache
1	21	,
1	22	die
1	23	Sperlinge
1	24	zwitscherten
1	25	und
1	26	tummelten
1	27	sich
1	28	dazwischen
1	29	;

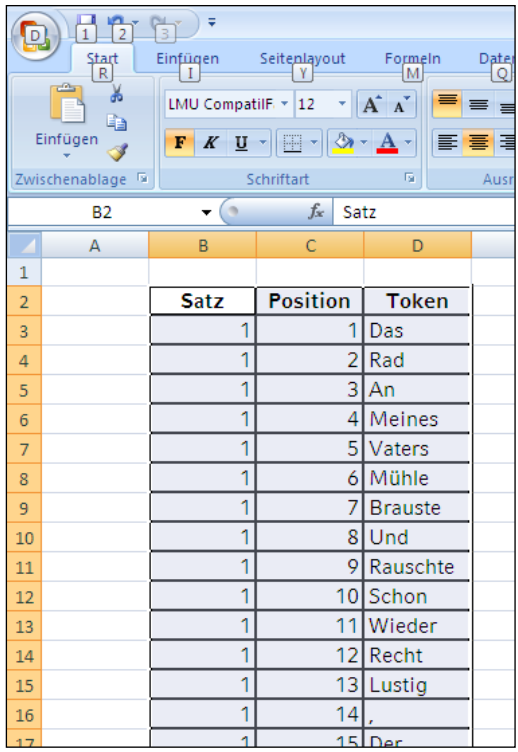
Die Kolumnen „Satz“ und „Position“ enthalten die Information darüber, in welchem Satz[41] und an welcher Position innerhalb des Satzes sich das jeweilige ↗Token in der dritten Kolumne befindet.

[41] Das Semikolon ist in diesem Fall als Satzbegrenzer definiert.

Der Import einer solchen ↗Tabelle in Excel kann auf verschiedene Weise erfolgen. Im Folgenden wird davon ausgegangen, dass die einzelnen Kolumnen in der Ausgangsdatei jeweils durch Tabulatorzeichen („\t“) von einander getrennt sind und dass es sich bei dieser Datei um eine reine Textdatei in ↗UTF-8-Kodierung handelt:

Abbildung 50: Eine csv-Datei mit farblich hervorgehobenen Tabulatorzeichen im Editor VIM

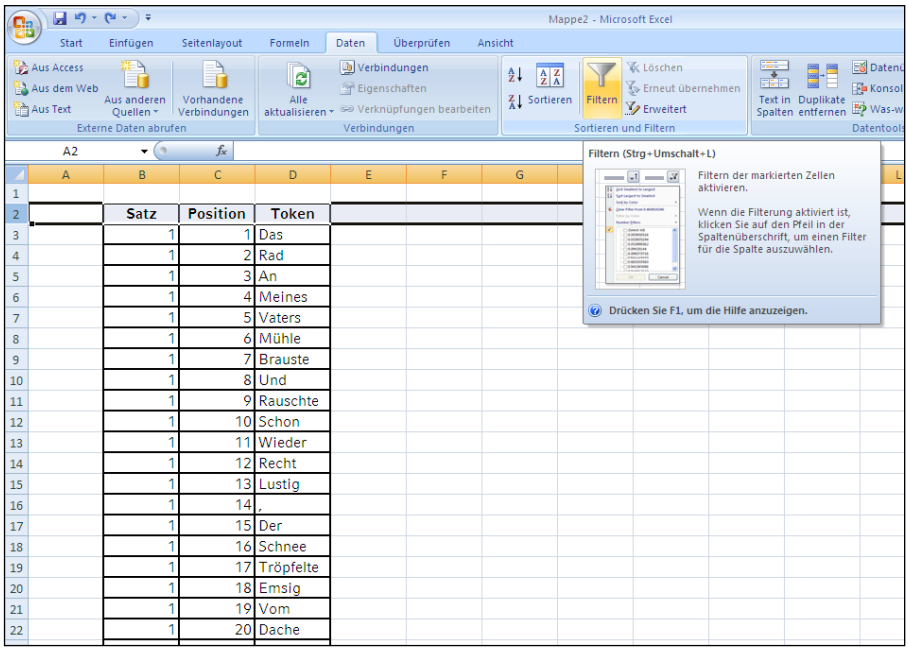
Die orange Markierung zeigt die Position der Tabulatorzeichen in der txt-Datei an (hier in VIM geöffnet). Durch eine einfache Copy-Paste-Operation kann dieser vorstrukturierte Text in eine Excel-Mappe importiert werden:



	A	B	C	D
1				
2		Satz	Position	Token
3		1	1	Das
4		1	2	Rad
5		1	3	An
6		1	4	Meines
7		1	5	Vaters
8		1	6	Mühle
9		1	7	Brauste
10		1	8	Und
11		1	9	Rauschte
12		1	10	Schon
13		1	11	Wieder
14		1	12	Recht
15		1	13	Lustig
16		1	14	,
17		1	15	Der

Abbildung 51: Eine Tabulator-strukturierte csv-Datei nach dem Einfügen in Excel 2007 mittels copy&paste

Die importierte ↗Tabelle läßt sich nun in eine sog. „Liste“ verwandeln. Dafür ist das Vorhandensein einer Kopfzeile unerlässlich. Diese Kopfzeile muss markiert werden. Anschließend wählt man im Menü „Daten“ das Symbol „Filtern“:



	Satz	Position	Token
3	1	1	Das
4	1	2	Rad
5	1	3	An
6	1	4	Melnes
7	1	5	Vaters
8	1	6	Mühle
9	1	7	Brauste
10	1	8	Und
11	1	9	Rauschte
12	1	10	Schon
13	1	11	Wieder
14	1	12	Recht
15	1	13	Lustig
16	1	14	,
17	1	15	Der
18	1	16	Schnee
19	1	17	Tröpfelte
20	1	18	Emsig
21	1	19	Vom
22	1	20	Dache

Abbildung 52: Generierung einer (sortier- und filterbaren) „Liste“ in Excel 2007

Nach Erstellung der Liste erscheinen neben den Einträgen in der Kopfzeile Schaltflächen mit kleinen, auf der Spitze stehenden Dreiecken. Ein Mausklick auf diese Schaltflächen öffnet ein Menü, das Filter- und Sortierfunktionen enthält:

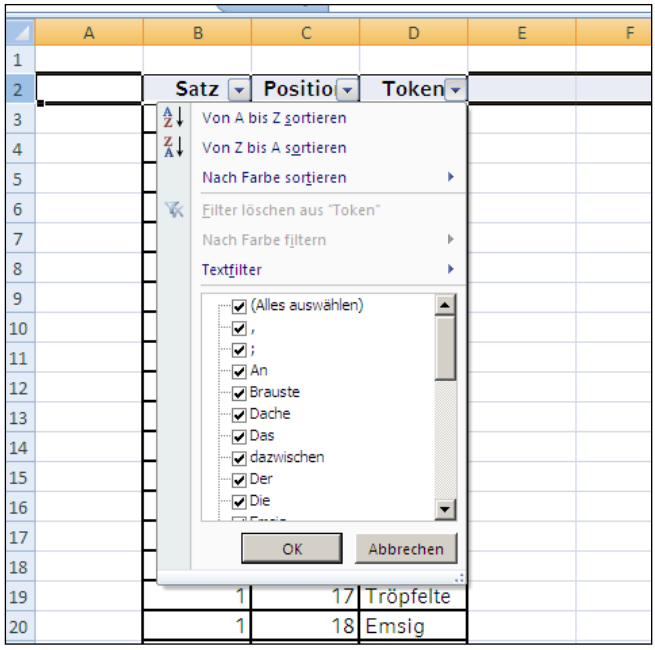
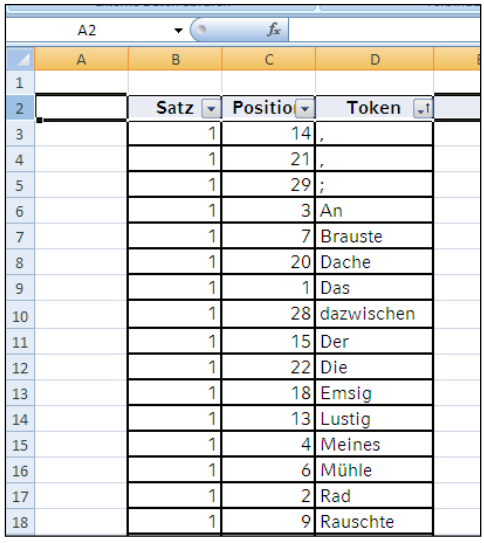


Abbildung 53: Menü mit Filter- und Sortierfunktionen in Excel 2007

Auf diese Weise läßt sich z.B. sehr einfach eine alphabetisch sortierte Tokenliste erzeugen:



Satz	Positio	Token
1	14	,
1	21	,
1	29	;
1	3	An
1	7	Brauste
1	20	Dache
1	1	Das
1	28	dazwischen
1	15	Der
1	22	Die
1	18	Emsig
1	13	Lustig
1	4	Meines
1	6	Mühle
1	2	Rad
1	9	Rauschte

Abbildung 54: Alphabetische sortierte Tokenliste in Excel 2007

Eine bestehende Liste kann nach rechts hin um weitere in die Liste integrierte Kolumnen erweitert werden. Das entsprechende Verfahren ist relativ einfach: Es wird in der Kopfzeile in die Zelle unmittelbar rechts neben dem letzten bestehenden Kolumnentitel ein neuer Titel eingetragen. Anschließend markiert man eine beliebige Zelle der Kopfzeile und ruft die Filterfunktion erneut auf. Excel erweitert dann die Liste um die neue Kolumne. Dies bietet sich z.B. für eine morphologische ↗Etikettierung der ↗Tokens an:

Satz	Position	Token	Wortart
1	1	Das	Artikel
1	2	Rad	Substantiv
1	3	an	Präposition
1	4	meines	Pronomen
1	5	Vaters	Substantiv
1	6	Mühle	Substantiv
1	7	brauste	Verb

Abbildung 55: Erweiterung einer Tokenliste in Excel 2007 um eine Kolumne mit Wortenartenetikettierung

Auf diese Liste können nun verschiedene Filteroperationen angewendet werden, wobei auch kolumnenübergreifend Filterkriterien miteinander kombiniert werden können. Das Kontextmenü jedes Kolumnentitels erlaubt eine einfache Filterung nach Maßgabe der jeweils in einer Kolumne enthaltenen Einträge.

Auf diese Weise lassen sich z.B. alle Sätze herausfiltern, die mit einem Artikel beginnen. Ein gesetzter Filter wird durch ein kleines Trichtersymbol angezeigt:

Satz	Positio	Token	Worta
1	1	Das	Artikel
1	15	Der	Artikel
1	22	Die	Artikel

Abbildung 56: Markierung gesetzter Filter durch ein Trichtersymbol in Excel 2007

Erweiterte Filteroptionen verbergen sich hinter dem Menüpunkt „Textfilter“

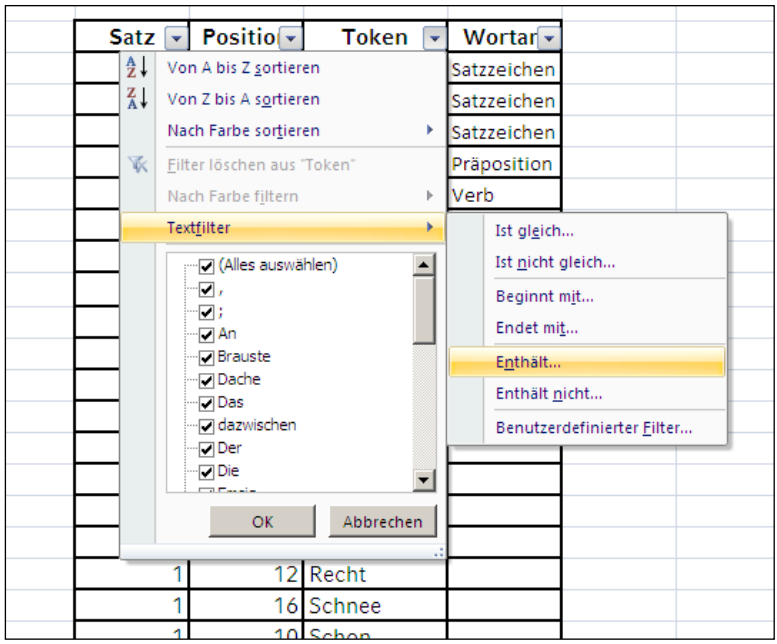


Abbildung 57: Erweiterte Filterfunktionen in Excel 2007

Der Menüpunkt „Benutzerdefinierter Filter“ erlaubt die Kombination von zwei Filterkriterien. Gesetzte Filter lassen sich durch den Menüpunkt „(Alle)“ wieder entfernen.

Neben der Sortierung und Filterung von Tabelleninhalten bietet Excel komfortable Möglichkeiten zu deren Analyse. Empfehlenswert ist vor allem die Nutzung der sog. „Pivot-Funktion“ („Pivot“: frz. „Dreh- und Angelpunkt“, mehr dazu auch unter http://de.wikipedia.org/wiki/Pivot-Tabelle).

Zur Erstellung einer Pivot-Analyse müssen zunächst alle Spalten markiert werden, deren Inhalte analysiert werden sollen. In aller Regel sollten das sämtliche Spalten sein, die Daten enthalten. Anschließend wählt man im Menüpunkt „Einfügen“ den Unterpunkt „Pivot-Tabelle“:

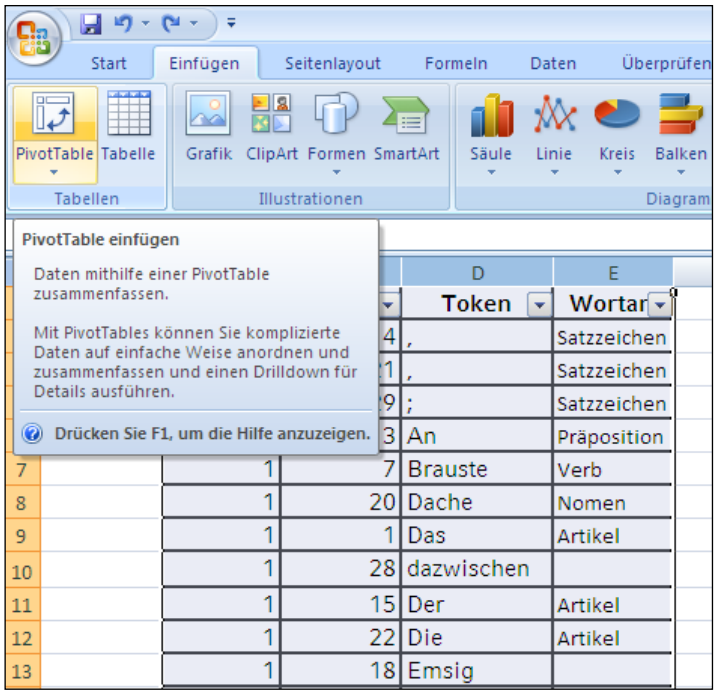


Abbildung 58: Erzeugung einer Pivot-Analyse in Excel 2007

Die folgende Dialogkette durchläuft man mit unveränderter Übernahme aller vorgeschlagenen Standardoptionen. Am Ende wird ein neues Tabellenblatt erzeugt, das folgendermaßen aussieht:

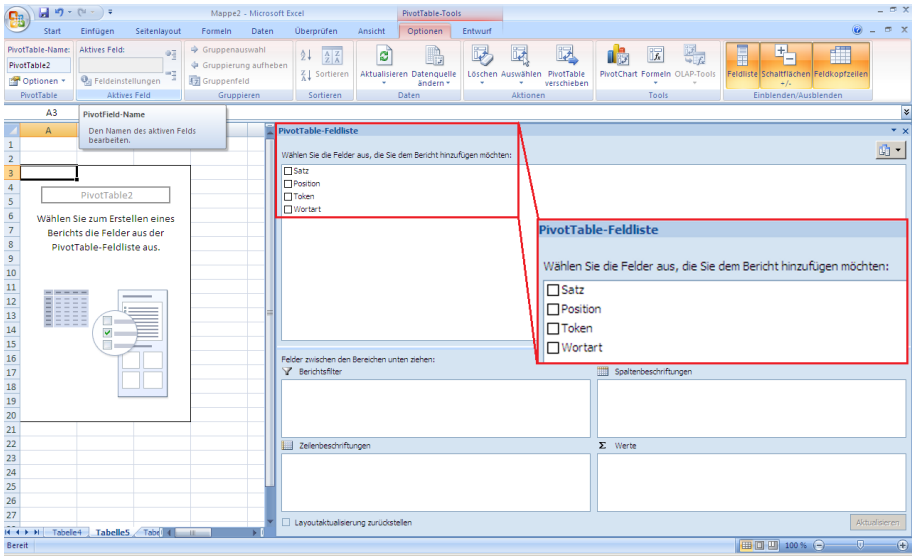


Abbildung 59: Tabellenblatt in Excel 2007 zur Erzeugung einer Pivot-Analyse

Durch Markieren der Felder im Bereich 1 und „Drag and Drop“ der markierten Felder in die Bereiche 2 bis 5 können unterschiedliche Analysen durchgeführt werden. So lassen sich in unserem Beispiel die ↗Tokens nach Maßgabe der zugewiesenen Wortarten gruppiert darstellen:

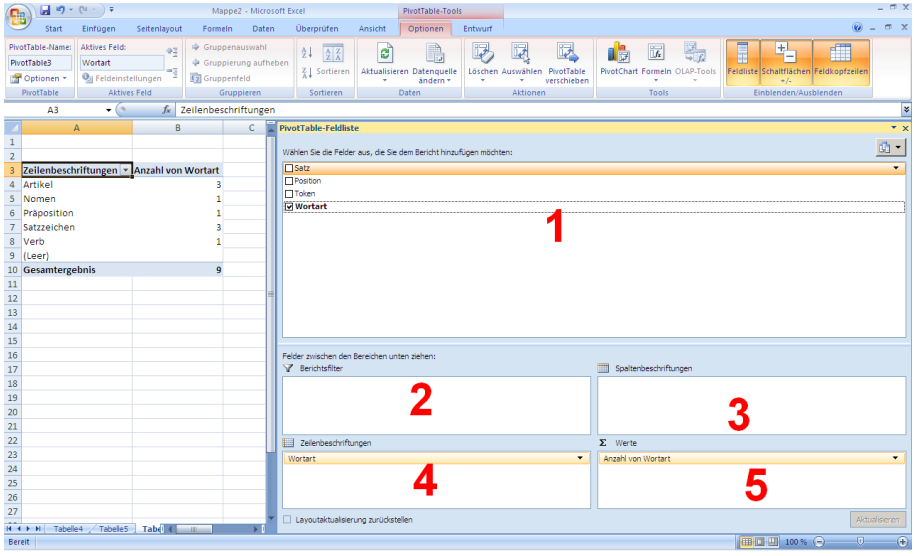


Abbildung 60: Funktionsfelder zur Erzeugung einer Pivot-Analyse in Excel 2007

Eine andere Möglichkeit bestünde darin, die einzelnen ↗Tokens den zugewiesenen Wortarten gegenüberzustellen. Zu diesem Zweck zieht man das Symbol „Token“ in den Bereich 4 („Zeilenbeschriftungen“) und das Symbol „Wortart“ in den Bereich 3 („Spaltenbeschriftungen“). Dies bewirkt, dass in der ↗Tabelle die Anzahl der ↗Tokens angezeigt wird, die der jeweiligen Wortart zugewiesen sind. Zusätzlich stehen wiederum Filterfunktionen zur Verfügung, die die Darstellung auf ausgewählte Kriterien reduzieren:

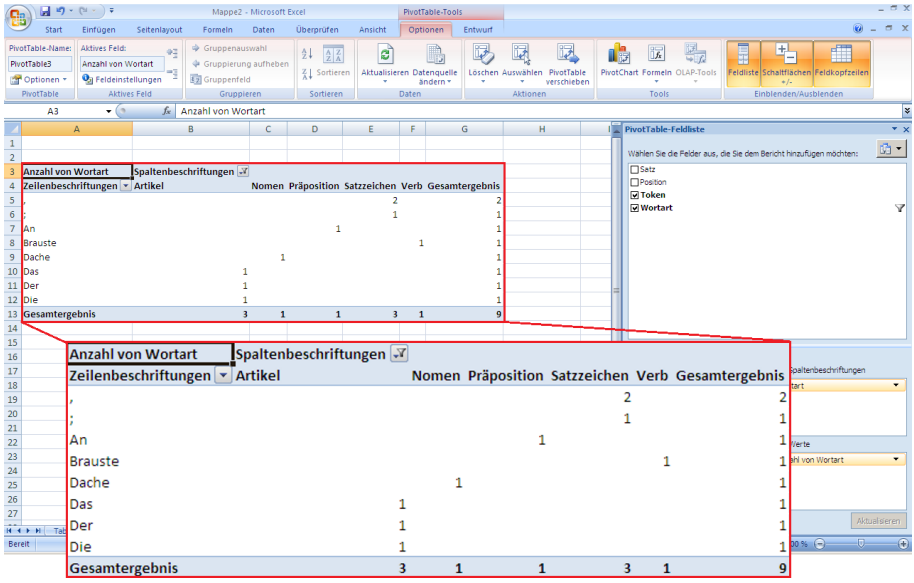


Abbildung 61: Datenfilterung innerhalb einer Pivot-Analyse in Excel 2007

Kriterienauswahl und -anordnung können jederzeit durch Drag- und Drop-Operationen verändert werden. Ein großer Vorteil der Pivot-Analyse ist, dass die Daten in der zugrundeliegenden ↗Tabelle stets unverändert bleiben.

Excel bietet hervorragende Möglichkeiten der Visualisierung von Daten bzw. Analyseergebnissen in Form von Diagrammen. Die Erstellung von Diagrammen erfolgt grundsätzlich in zwei Schritten: Zunächst muss ein Datenbereich in einer ↗Tabelle ausgewählt werden. Anschließend ruft man über das Menü „Einfügen“ den Unterpunkt „Diagramm ... “ auf.

Die Diagrammfunktion ist besonders sinnvoll bei Anwendung auf die Ergebnisse einer Pivot-Analyse. So läßt sich beispielsweise die Gruppie-

rung der ↗Tokens nach Maßgabe der zugewiesenen Wortarten mit wenigen Klicks z.B. in Gestalt eines Tortendiagramms darstellen:

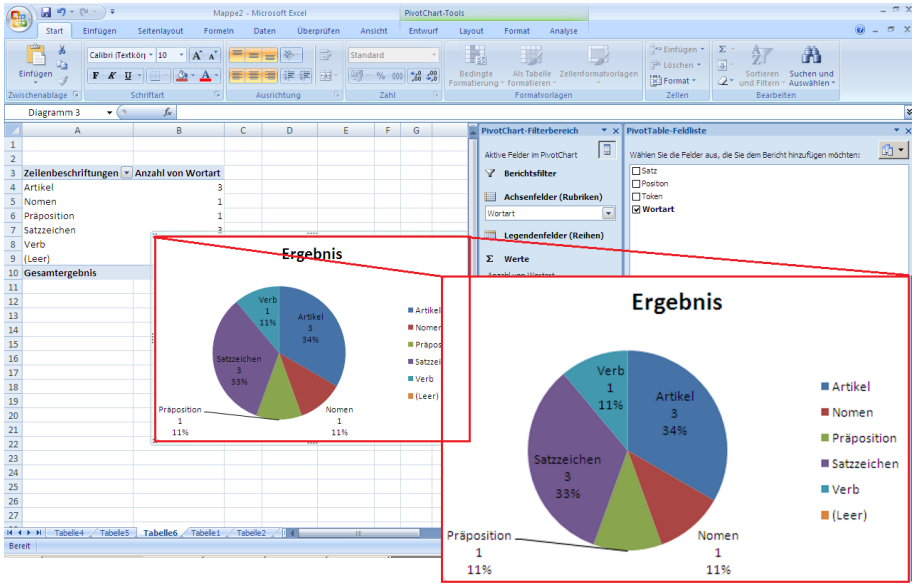


Abbildung 62: Visualisierung einer Pivot-Analyse (Häufigkeitsverteilung von Wortarten) in Gestalt eines Tortendiagramms in Excel 2007

5.2 Relationale Datenbanken / SQL

Es gibt unterschiedliche Arten von Datenbanksystemen. Im Folgenden ist ausschließlich von sog. relationalen ↗Datenbanken die Rede, die sich im Einsatz für die Korpuslinguistik sehr gut bewährt haben.

Relationale ↗Datenbanken sind im Kern eigentlich nichts anderes als eine Sammlung von einer oder – meistens – mehreren ↗Tabellen. Der eigentliche Mehrwert einer ↗Datenbank besteht in ihrer Fähigkeit, das Material in den ↗Tabellen mit großer Geschwindigkeit durchsuchen und bei dieser Suche die Inhalte der verschiedenen ↗Tabellen nach vorgegebenen Kriterien miteinander verknüpfen zu können.

Für die Arbeit mit ↗Datenbanken benötigt man sog. Datenbankverwaltungssysteme (auch: „Datenbankmanagementsysteme“). Ein sehr bekanntes Datenbankverwaltungssystem ist z.B. Microsoft Access. Der Unterschied zwischen Datenbankverwaltungssystem und Datenbankobjekten läßt sich vielleicht so illustrieren:

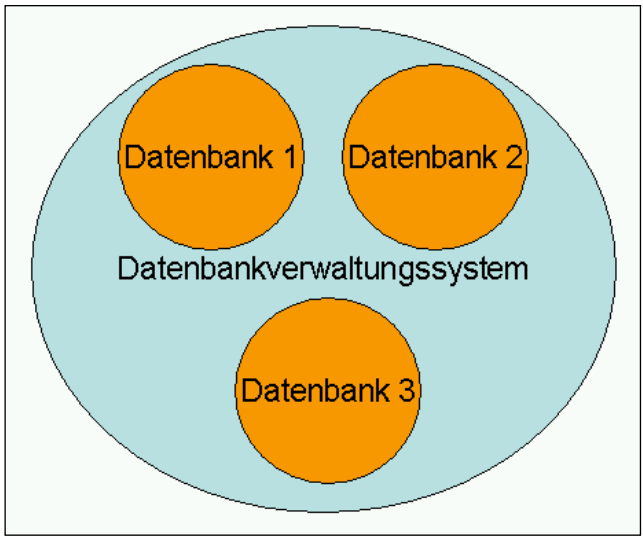


Abbildung 63: Datenbankverwaltungssystem und darin enthaltene ↗Datenbanken

Für den Einsatz in der Korpuslinguistik eignet sich das Datenbankverwaltungssystem MySQL sehr gut. Es ist kostenlos für verschiedene ↗Betriebssysteme verfügbar, es ist „Open Source", es ist zuverlässig und schnell und es ist dazu geeignet, mit einem sog. Webserver verknüpft zu werden, so dass die gespeicherten Daten in Webseiten integriert werden können.

MySQL besitzt im wesentlichen drei „Benutzerschnittstellen", also Methoden, wie der Benutzer das System bedienen kann: MySQL kann von der Kommandozeile eines ↗Terminals / einer ↗Shell angesprochen werden, ferner besteht die Möglichkeit, die ↗Datenbank über eine Webseite (phpMyAdmin) zu bedienen und schließlich kann man noch eines Clientprogramms verwenden.[42]

5.2.1 MySQL und phpMyAdmin (PMA)

5.2.1.1 phpMyAdmin: Einstellungen und Grundfunktionen

Die wohl populärste Benutzeroberfläche für die Verwendung von MySQL-↗Datenbanken ist phpMyAdmin (PMA). PMA ist kostenfrei und kann unter www.phpmyadmin.net heruntergeladen werden. Installation und vor allem die Wartung verlangen jedoch vertiefte Kenntnisse auf

[42] Für Windows-Systeme erscheint uns das Programm HeidiSQL (https://www.heidisql.com/) besonders empfehlenswert.

dem Gebiet der Client-/Serveradministration. Die IT-Gruppe Geisteswissenschaften der LMU stellt daher den Studierenden und Promovierenden im Bereich der Geisteswissenschaften an der LMU eine zentrale MySQL-/PhpMyAdmin-Installation zur Verfügung.

Das Login erfolgt mit dem von der ITG vergebenen Benutzernamen und Passwort über die folgende ↗URL: https://pma.gwi.uni-muenchen.de:8888/

Nach der Auswahl eines Servers (Dialogfeld "Aktueller Server") erscheint folgende Webseite:

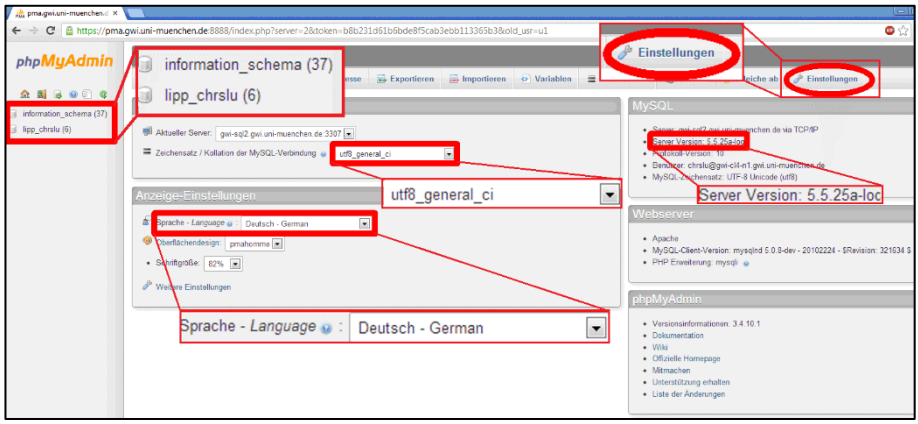


Abbildung 64: Startseite von PMA nach dem Einloggen

Am linken Rand sind die ↗Datenbanken aufgelistet, auf die der Benutzer Zugriffsrechte besitzt. Als ↗Zeichensatz bzw. ↗Kollation der MySQL-Verbindung muss ein Vertreter der ↗UTF-8-Familie ausgewählt werden. In Betracht kommen normalerweise utf8_bin, utf8_general_ci und utf8_unicode_ci. Die ↗Kollation beeinflußt das Ergebnis von Sortierungen. „ci“ steht für „case-insensitive“, d.h. diese ↗Kollationen unterscheiden – anders als utf8_bin – nicht zwischen Groß- und Kleinschreibung.[43] Grundsätzlich ist zu empfehlen, hinsichtlich der ↗Kollationen möglichst homogen zu verfahren: Die ↗Kollation der MySQL-Verbindung sollte mit den – wiederum einheitlichen – ↗Kollationen der einzelnen Datenbanktabellen übereinstimmen.

[43] Die ↗Kollationen utf8_general_ci und utf8_unicode_ci unterscheiden sich nur marginal. utf8_unicode_ci ist jünger als utf8_general_ci und ist tendenziell "regelkonformer" (so gilt für utf8_unicode_ci z.B. ß=ss, während für utf8_general_ci ß=s gilt), dafür sind Sortierungen mittels utf8_general_ci angeblich schneller.

Die PMA-Startseite liefert außerdem Informationen zur installierten MySQL-Version (hier: 5.5.25a) und gibt die Möglichkeit, die Sprache der Benutzeroberfläche einzustellen.

Über den Reiter „Einstellungen“ (rechts oben) lassen sich sodann weitere individuelle Optionen festlegen. Die meisten der dort möglichen Anpassungen sind selbsterklärend. Hingewiesen sei an dieser Stelle nur auf die Option Einstellungen ⇒ Hauptframe ⇒ Anzeigemodus ⇒ Tabellenbearbeitung mittels Icons, wo der Wert „Ja“ angewählt werden sollte, um die ↗Tabellen im Bearbeitungsmodus möglichst übersichtlich gestaltet zu bekommen:

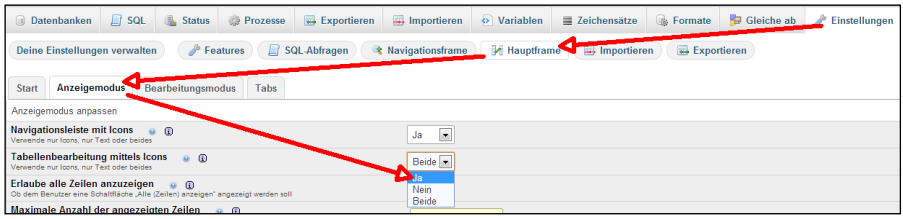


Abbildung 65: Festlegung individueller Einstellungen in PMA

Empfohlen sei außerdem die Überprüfung der Einstellungen unter „Importieren“ und „Exportieren“, die sich auf den Im- und Export von Daten in die bzw. aus der ↗Datenbank beziehen.

Die Arbeit mit der eigentlichen ↗Datenbank (in unserem Beispiel: lipp_chrslu) beginnt mit dem Klick auf den entsprechenden Listeneintrag am linken Fensterrand:

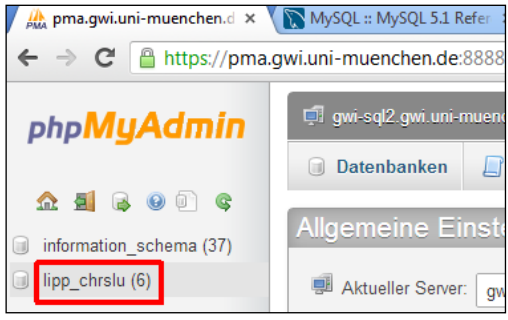


Abbildung 66: Auswahl einer ↗Datenbank in PMA

Anschließend werden die in der ↗Datenbank vorhandenen ↗Tabellen gelistet, die wiederum durch Anklicken ausgewählt werden können:

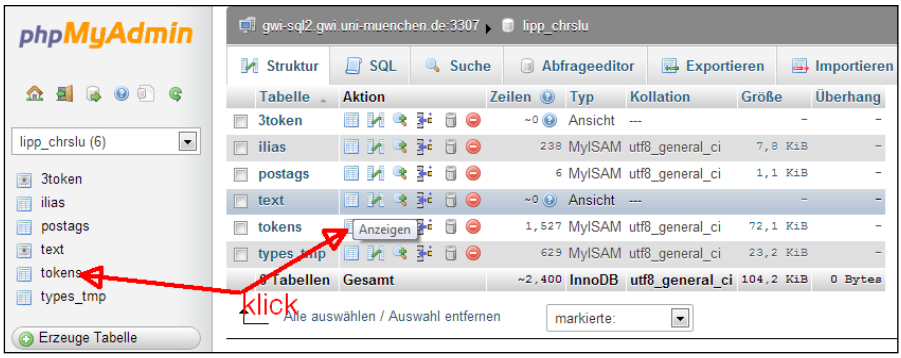


Abbildung 67: Auswahl einer ↗Tabelle in PMA

5.2.1.2 Import von Dat(ei)en in eine bestehende Datenbank

Der Import von Daten aus einer Datei erfolgt über den Reiter „Importieren“. Wesentliche Voraussetzung ist, dass die Daten in der Datei

- konsistent kodiert (Empfehlung: ↗UTF-8),
- durch Zeilenumbrüche in Datensätze gegliedert und diese
- durch Verwendung von eindeutigen ↗Separatoren (Empfehlung: \t [= Tabulator]) in einzelne Felder zerlegt sind, wobei in jeder ↗Zeile stets dieselbe Anzahl von Feldern vorliegen muss.

Der Import erfolgt dann über die Auswahl der entsprechenden Datei und die Festlegung bestimmter Parameter, wie der in der Datei verwendeten Kodierung, der in der Datei vorliegenden Datenstruktur (csv – „character separated values“), des verwendeten ↗Separators sowie der Information, ob die Werte in den Spalten von einem bestimmten Zeichen (standard: ") eingeschlossen sind. Sofern zutreffend, kann angegeben werden, dass die erste ↗Zeile der Datei die Namen der Spalten enthält (empfohlen!).

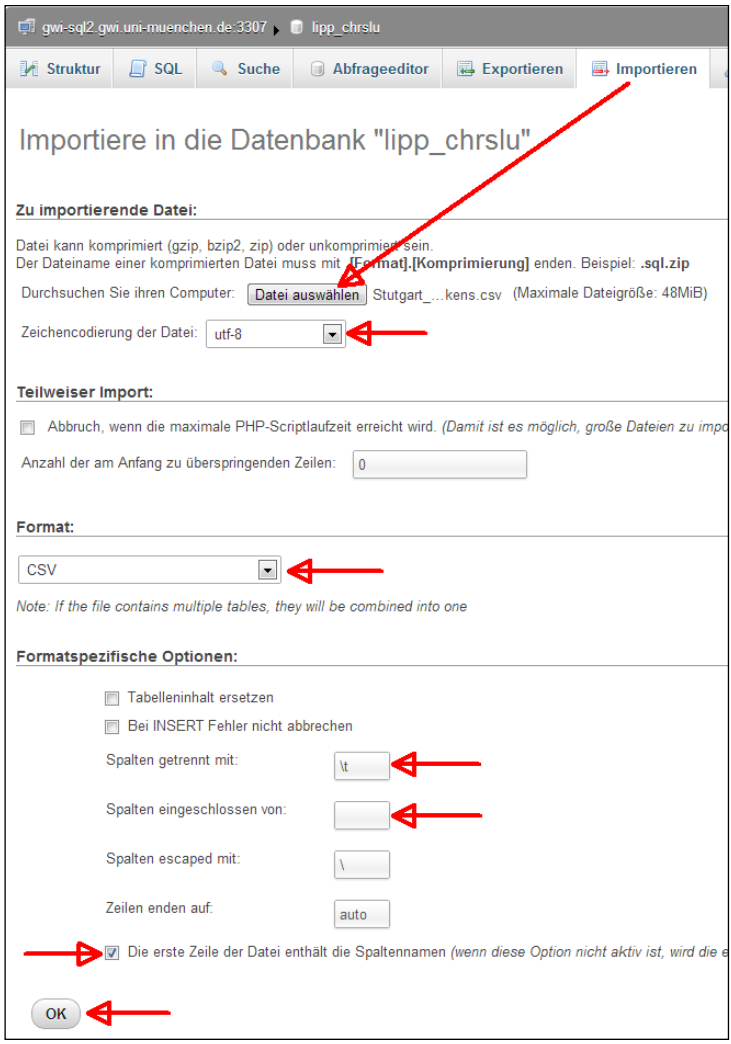


Abbildung 68: PMA-Dialog zum Import einer csv-Datei in eine ↗Datenbank

Wichtig ist ferner, dass man sich beim Import auf der obersten Ebene der ↗Datenbank befindet und keine ↗Tabelle geöffnet hat. Andernfalls

versucht MySQL, die importierten Daten in die ausgewählte ↗Tabelle zu importieren.[44]

Der erfolgreiche Import wird von MySQL mit einer entsprechenden Meldung signalisiert. Anschließend erscheint in der Liste am linken Fensterrand eine neue ↗Tabelle mit dem generischen Namen `table n` (n = eine natürliche Zahl):

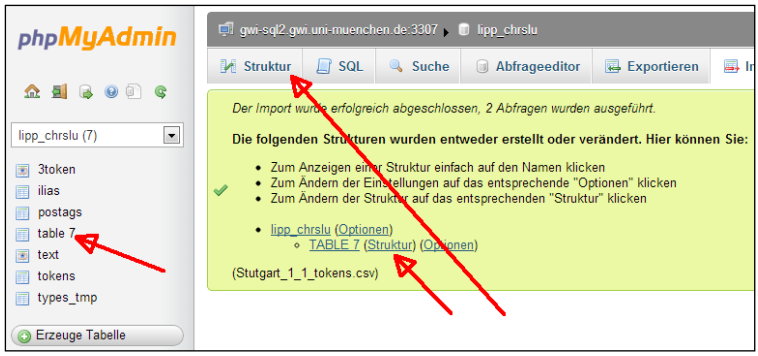


Abbildung 69: PMA nach erfolgreichem Datenimport einer csv-Datei

Bei der Anlage der ↗Tabelle definiert und dimensioniert MySQL die einzelnen Felder nach Maßgabe der aktuell importierten Daten. Es ist im Hinblick auf unter Umständen später in die ↗Tabelle einzutragende Daten unerläßlich, diese Einstellungen zu überprüfen und ggf. zu korrigieren. Nach Klick auf „Struktur“ zeigt sich folgendes Bild:

[44] Häufig ist die Größe von Dateien, die mittels PMA in der beschriebenen Weise in eine ↗Datenbank importiert werden können, durch entsprechende Eintragungen in die serverseitigen Konfigurationsdateien von PMA beschränkt. Beim Versuch, zu große Dateien zu importieren, erscheint eine Fehlermeldung. Diesem Problem kann begegnet werden, indem man die zu importierende Datei in eine Anzahl kleinerer Teildateien zerlegt. Zur Zerlegung kann man z.B. das Unix-Tool „split“ verwenden (s. die Beschreibung oben S. 76)

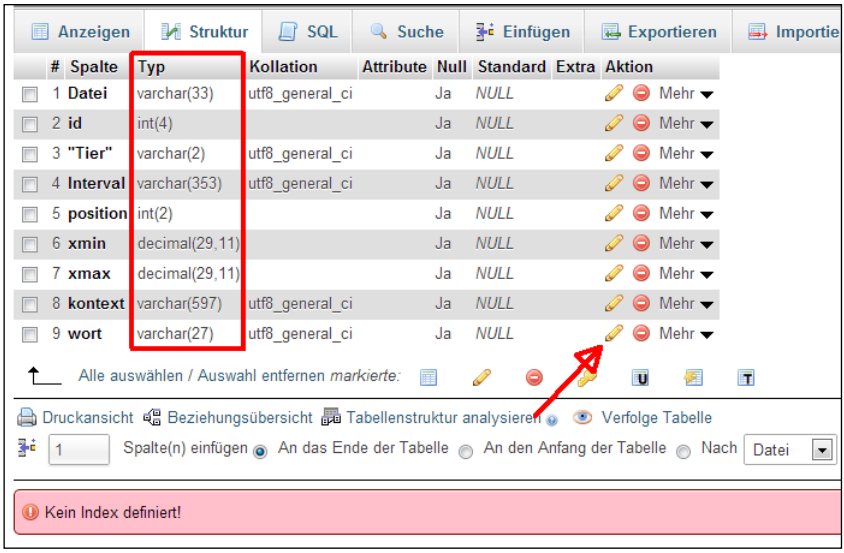


Abbildung 70: Automatisch von PMA beim Datenimport generierte Struktur der neuen ↗Tabelle mit Angabe der einzelnen Felddefinitionen

Im vorliegenden Beispiel wurde das Feld `wort` als Datentyp „varchar“ („various characters“) mit einer Länge von maximal 27 Zeichen definiert. Dies bedeutet, dass alle Daten, die später in dieses Feld eingetragen werden, dem Datentyp „varchar“ entsprechen müssen und nicht länger als 27 Zeichen sein dürfen. Längere Daten würden hinter dem 27sten Zeichen einfach abgeschnitten.

Die Modifizierung der Felddefinition erfolgt durch Klick auf das Stiftsymbol. Speziell bei der Dimensionierung empfiehlt sich Großzügigkeit. Die Umbenennung der ↗Tabelle in einen sinnvollen Namen schließlich erfolgt über den Reiter Operationen:

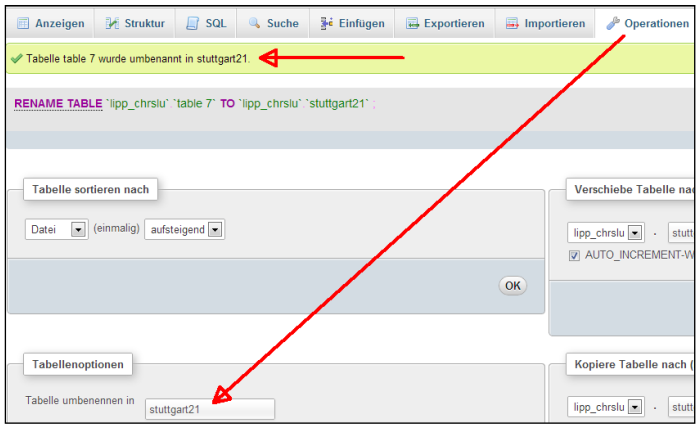


Abbildung 71: Umbenennung einer ↗Tabelle in PMA

Anschließend sind der Datenimport und die Anpassung der Strukturdefinitionen nach dem Datenimport abgeschlossen, und die neue ↗Tabelle ist bereit für Analysen und Bearbeitungen. Ein Klick auf den Tabellennamen am linken Fensterrand führt zur Anzeige der mit den Daten gefüllten ↗Tabelle:

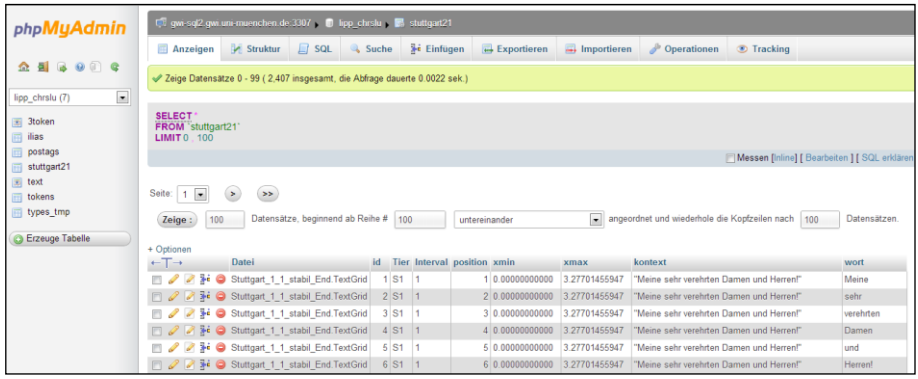


Abbildung 72: Anzeige der „fertigen“ ↗Tabelle in PMA nach Import aus csv-Datei

Daten können auch manuell über die Eingabemaske, die sich hinter dem Reiter „Einfügen“ verbirgt, eingegeben werden.

5.2.1.3 Datenfilterung und -analyse

Das PMA-Interface bietet für die Datenanalyse vorgefertigte Abfrageformulare, die den Einstieg in die spezielle Abfragesyntax (↗SQL) erleichtern. Auf der obersten Datenbankebene, d.h. vor Auswahl einer bestimmten ↗Tabelle, steht zu diesem Zweck der Reiter „Abfrageeditor“

zur Verfügung, nach Auswahl einer ↗Tabelle findet sich eine Abfragemaske unter dem Reiter „Suche“. Einen vollgültigen Ersatz für das Erlernen von ↗SQL stellen jedoch weder der Abfrageeditor noch die Suche-Maske dar. Abgesehen davon, dass der Umgang mit den speziellen PMA-Funktionen mindestens so komplex wie ↗SQL selbst ist, bieten die vorgefertigten Abfragemodule bei weitem nicht alle Analysemöglichkeiten, die mit ↗SQL realisiert werden können. Es erscheint daher vernünftiger, zumindest mittelfristig den souveränen Umgang mit ↗SQL zu erlernen. Für den Anfang bzw. für einfache Abfragen ist die Verwendung der vorgefertigten Module jedoch durchaus vernünftig. Folgende Abbildung zeigt die Abfragemaske für die ↗Tabelle „stuttgart21“:

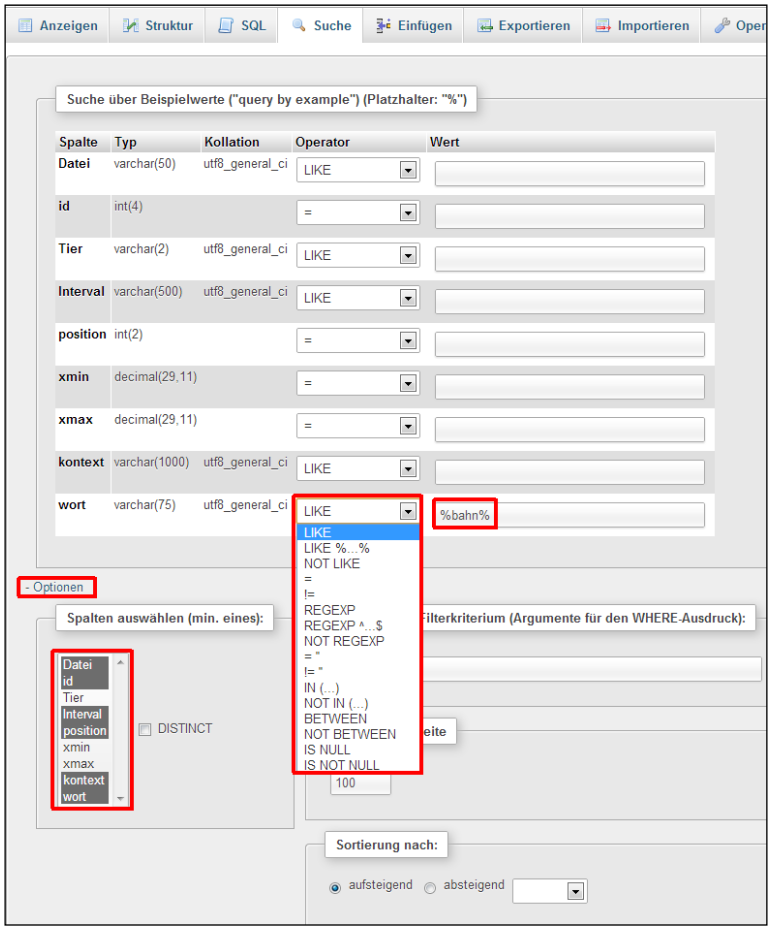


Abbildung 73: Suchmaske in PMA

Die Suchmaske listet die einzelnen Spalten der ↗Tabelle untereinander auf. Im Feld „Wert“ können die Suchkriterien eingegeben werden. Wichtig ist die Auswahl des sog. Such-Operators. Bei Feldern des Typs „varchar“ ist in PMA standardmäßig der Operator „LIKE“ ausgewählt. „LIKE“ erlaubt die sog. Trunkierung durch Verwendung spezieller „Wildcards“: Das Prozentzeichen (%) steht für eine beliebige Anzahl beliebiger Zeichen, der Unterstrich (_) für genau ein beliebiges Zeichen. Die Syntax von ↗SQL lässt bei der Formulierung von Anfragen Groß- und Kleinschreibung zu. Über die Option „Spalten auswählen“ läßt sich festlegen, ob bei der Präsentation des Suchergebnisses alle oder nur ausgewählte Spalten der ↗Tabelle angezeigt werden sollen. Das oben gegebene Beispiel führt zur Auswahl der Spalten `id`, `Interval`, `position` und `wort` all der Datensätze der ↗Tabelle `stuttgart21`, die im Feld `wort` den ↗String „bahn“ aufweisen, wobei „bahn“ aufgrund der Verwendung des Operators „like“ und der Wildcards „%“ am Anfang, in der Mitte oder am Ende des Eintrags stehen kann:

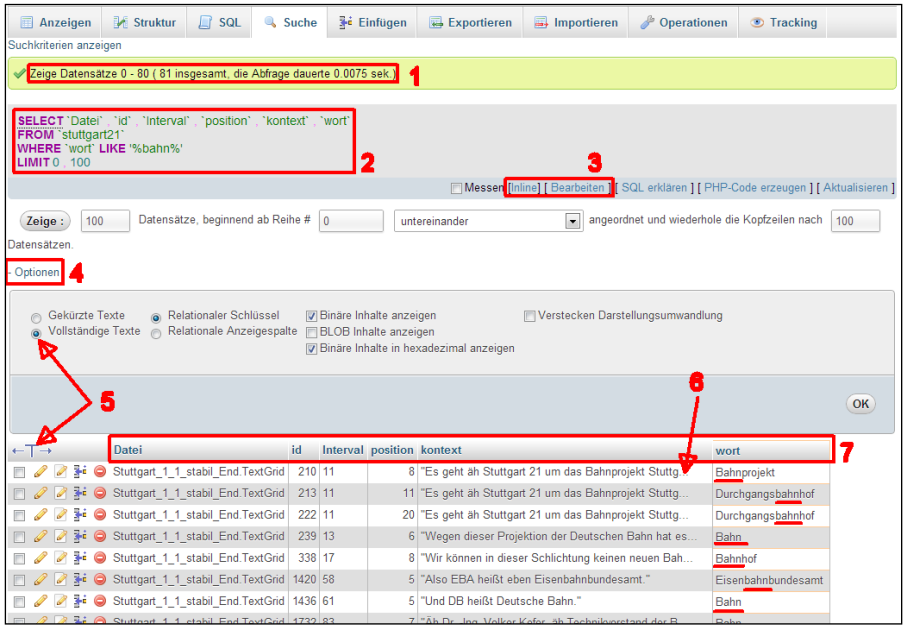


Abbildung 74: Ergebnis einer Datensuche unter Verwendung der PMA-Suchmaske

Im grünen Feld zeigt PMA die Anzahl der gefundenen Datensätze (1), nämlich 81. In der Grundeinstellung begrenzt PMA die Ausgabe von längeren Feldinhalten (erkennbar an den drei Punkten [6]). Die Ausgabe der vollständigen Feldinhalte kann entweder durch Klick auf das große

T oder über Optionen (4) ⇒ Vollständige Texte ⇒ OK bewirkt werden (5).

Das abgebildete Ergebnis zeigt, dass die Suche keine Rücksicht auf die Groß-/Kleinschreibung von Daten in der ↗Tabelle nimmt. Ist dies gewünscht, muss die Suche unter Anwendung einer „case-sensitven" ↗Kollation, z.B. utf8_bin, erfolgen. Für diesen Zweck kennt ↗SQL die Anweisung „collate utf8_bin", die in die Abfragesyntax integriert werden muss. Am einfachsten ist dies möglich, indem man die dem Abfrageergebnis zugrundeliegende ↗SQL-Anweisung (2) modifiziert. Dies geschieht durch Klick auf „Inline" bzw. „Bearbeiten" (letzteres öffnet ein neues Browserfenster; ansonsten identisch mit „Inline") (3).

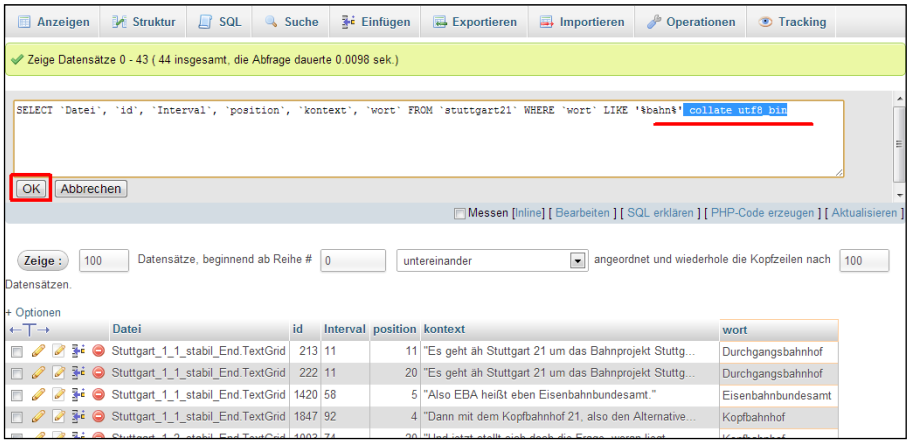


Abbildung 75: Modifizierung eines Suchstatements in PMA

Eine Sortierung des Ergebnisses ist am einfachsten durch Klick auf die Kolumnentitel der Ergebnistabelle möglich (s. Abbildung 74 [7]). Beim ersten Klick erfolgt die Sortierung aufsteigend, bei neuerlichem Klick wird die Sortierung umgekehrt.

5.2.1.4 Direkteingabe von SQL-Befehlen

Der Reiter „SQL" erlaubt die direkte Eingabe von ↗SQL-Befehlen und eröffnet somit Möglichkeiten, die weit über die Funktionalität der in PMA eingebauten Suchmasken hinausgeht:

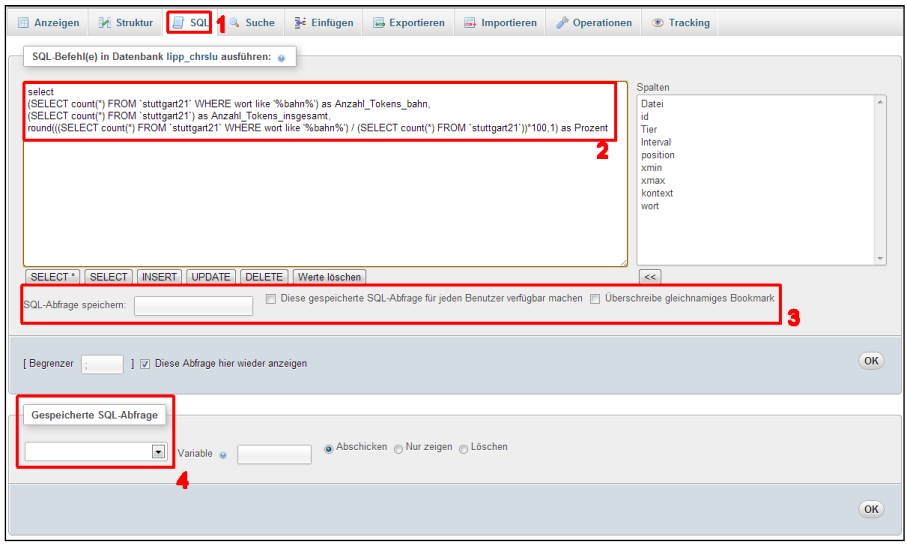


Abbildung 76: Direkteingabe von ↗SQL-Statements in der ↗SQL-Maske von PMA

Neben der Eingabe quasi beliebiger ↗SQL-Abfragen und -Kommandos können Abfragen hier auch für spätere Wiederverwendung gespeichert werden (3). Solche gespeicherten Abfragen können über den Punkt (4) aufgerufen, verändert oder auch wieder gelöscht werden. Es ist jedoch empfehlenswert, das persönliche Repertoire immer wieder benötigter ↗SQL-Anweisungen in einer Textdatei auf seinem Client zu verwalten und bei Bedarf Befehle mittels Copy/Paste in das ↗SQL-Feld von PMA zu übertragen und ausführen zu lassen.

Eine einmal ausgeführte Anweisung kann anschließend über den Menüpunkt „Inline“ oder „Bearbeiten“ modifiziert und erneut ausgeführt werden (s. auch oben Abbildung 74 [3]):

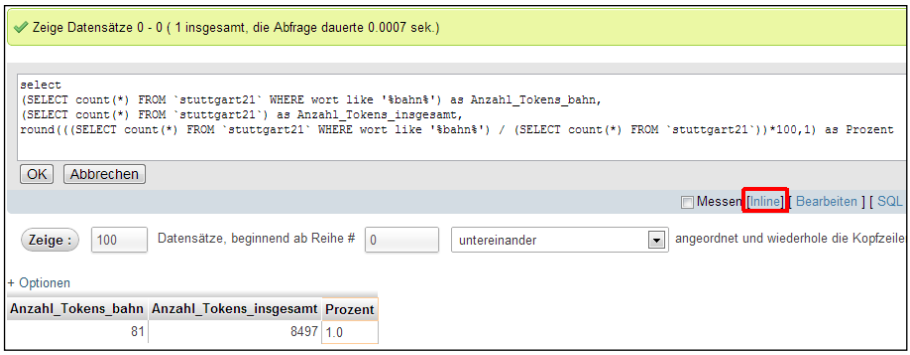


Abbildung 77: Abermalige Modifizierung eines ↗SQL-Statements

5.2.1.5 Datenexport und -sicherung

Der Reiter „Exportieren“ gestattet den Datenexport in einer ganzen Reihe von Formaten. So können die Daten z.B. als CSV-Dateien oder im XML-Format ausgelesen werden. Für die Anfertigung einer Sicherheitskopie der gesamten ↗Datenbank ist es empfehlenswert, von Zeit zu Zeit einen vollständigen Export im ↗SQL-Format vorzunehmen und die erzeugte Textdatei auf einem lokalen Datenträger zu sichern.

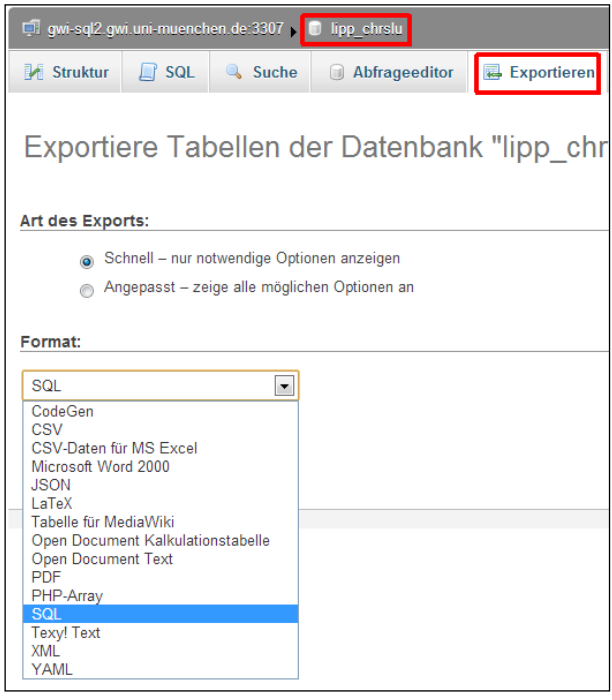


Abbildung 78: Dialogmaske von PMA zum Export von Daten aus einer ↗Datenbank

Die vollständige ↗Datenbank wird nur exportiert, wenn keine ↗Tabelle ausgewählt ist. Andernfalls werden nur die Daten der ausgewählten ↗Tabelle exportiert!

Für die Wiederherstellung einer verlorenen oder korrupten ↗Datenbank reicht es aus, die ↗SQL-Exportdatei mit dem vollständigen Datenbank-Export über den Reiter „Importieren“ wieder einzulesen.

Ergebnisse von Suchabfragen können über die Option „Exportieren“ im Bereich „Operationen für das Abfrageergebnis“ unterhalb der Ergebnistabelle ausgelesen werden:

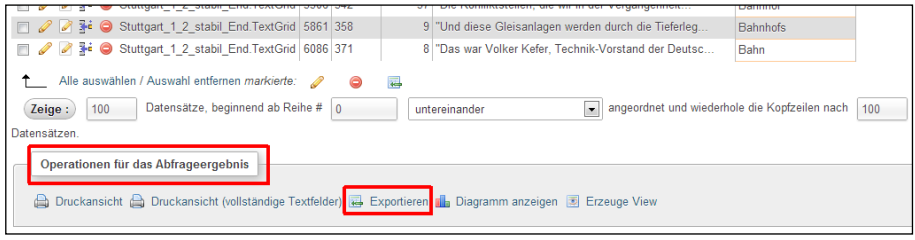


Abbildung 79: Export von gefilterten Daten

5.2.1.6 Erzeugung von Indizes

Für die Beschleunigung von Abfragen ist die Definition von Indizes unerläßlich. Einen Datenbankindex kann man sich ganz analog zu einem Index in einem herkömmlichen Buch vorstellen: Es handelt sich um eine nach Zahlen oder Buchstaben sortierte Liste sämtlicher Einträge in einem (oder mehreren) Tabellenfeld(ern), verknüpft mit der Information, an welcher Stelle innerhalb einer ↗Tabelle sich der entsprechende Datensatz befindet. MySQL unterscheidet mehrere Arten von Indizes, und PMA erlaubt deren bequeme Erzeugung über den Reiter „Struktur“ einer Datenbanktabelle:

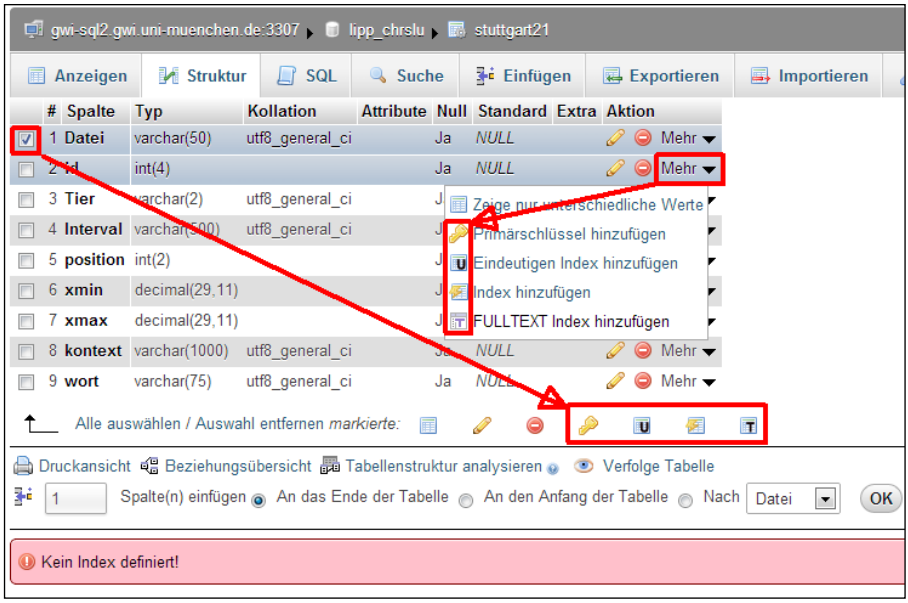


Abbildung 80: Erzeugung von Indizes in PMA

Die Anlage eines Index' erfolgt entweder durch Ankreuzen eines oder mehrerer Felder am Anfang eines Listeneintrags oder – wenn ein Index lediglich über ein einziges Feld angelegt werden soll – über das über „Mehr“ erreichbare Kontextmenü. Die verschiedenen Indizes haben unterschiedliche Eigenschaften und Funktionen:

- Ein „Primary Key“ (Primärschlüssel) enthält normalerweise eine Liste eindeutiger Identifikatoren („ID“) in Gestalt von positiven Ganzzahlen. Es dürfen keine Duplikate auftreten, und meist ist mit dem Feld des Primary Key eine „Autoincrement“-Funktion verknüpft, die bewirkt, dass jeder neu hinzugefügte Datensatz automatisch in diesem Feld die jeweils nächst höhere Ganzzahl zugewiesen bekommt.
- „Index“ erzeugt eine sortierte Liste aller Werte in einem Feld. Im Gegensatz zum Primary Key sind dabei Duplikate erlaubt. Ein solcher Index eignet sich z.B. für Felder, die Einzelwörter bzw. ↗Tokens enthalten.
- Während „Index“ den Inhalt eines jeden Feldes jeweils nur als Ganzes behandelt, zerlegt „FULLTEXT Index“ die Feldinhalte in Einzeltokens und erzeugt daraus einen Index. Diese Index-Art

eignet sich demnach am besten für Felder, die Textpassagen enthalten.

- Ein „Unique-Index“ ist im Kern nichts anderes als ein „Index“, allerdings mit der zusätzlichen Bedingung, dass keine Duplikate auftreten dürfen. Insofern ist er identisch mit einem Primary Key, jedoch mit dem Unterschied, dass für eine ↗Tabelle jeweils nur ein einziger Primary Key definiert werden kann. Hingegen ist es möglich, eine im Prinzip beliebige Anzahl von Unique-Indizes anzulegen.

Jeder der genannten Indizes kann über mehrere Felder einer Datenbanktabelle angelegt werden. Ein solcher kombinierter Index ist vor allem dann erforderlich, wenn Tabellenverknüpfungen durch den Vergleich der Inhalte von jeweils mehr als einem Feld erfolgen. Ein Beispiel aus dem AsiCa-Korpus: Die beiden ↗Tabellen `wort` (=↗Tokens) und `intervall` hängen in logischer Hinsicht dergestalt zusammen, dass die in beiden ↗Tabellen vorhandenen Felder `interview` und `intervall` jeweils korrespondierende Werte enthalten. Zur entsprechenden Verknüpfung der beiden ↗Tabellen verwendet man folgende ↗SQL-Abfrage:

```
select *
from wort w
join intervall i using (interview, interval);
```

Je nach Anzahl der Datensätze in den ↗Tabellen, kann diese Abfrage unter Umständen sehr lange dauern. Zur Beschleunigung müßte in diesem Fall in beiden ↗Tabellen ein Index angelegt werden, der die beiden Felder `interview` und `intervall` miteinander kombiniert. Dies erfolgt entweder über den Reiter ‚Struktur‘ in PMA, indem die beiden Felder angehakt werden und anschließend auf das Index-Symbol geklickt wird:[45]

[45] Alternativ kann man folgendes ↗SQL-Statement verwenden: `ALTER TABLE `wort` ADD INDEX ( `Interview` , `Intervall` );`

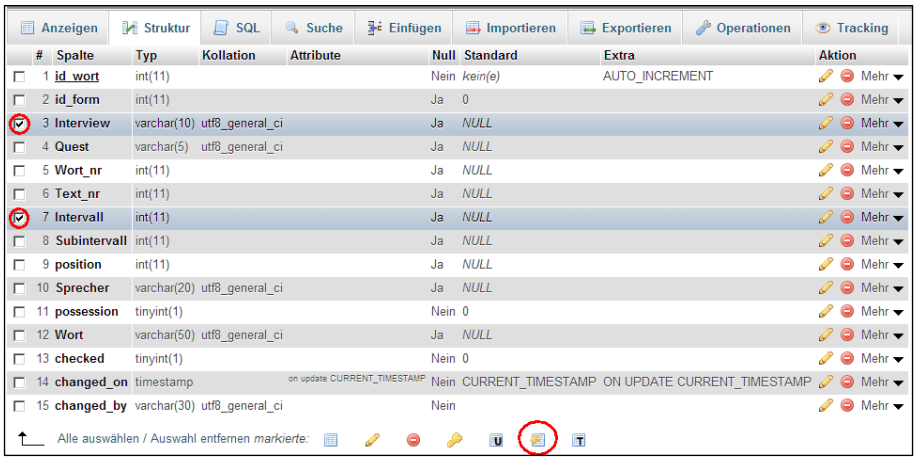


Abbildung 81: Erzeugung von Indizes, die mehrere Felder umfassen, in PMA

Indizes können jederzeit gelöscht und neu erzeugt werden. Es ist unbedingt darauf zu achten, dass nicht mehr Indizes erzeugt werden als nötig sind. Vor allem sollte die Anlage von mit einander konkurrierenden Indizes vermieden werden. Es empfiehlt sich, Indices dann neu anzulegen, nachdem ein Datenimport oder Datenreload durchgeführt worden ist.

5.2.2 SQL – Structured Query Language

Um das Potential einer MySQL-↗Datenbank voll auszuschöpfen, ist das Erlernen der spezifischen Abfragesprache ↗SQL unerläßlich. Bei Nutzung des Interface PMA ist die Ausführung von ↗SQL-Befehlen über die Eingabemaske „SQL“ möglich. Wir konzentrieren uns im Folgenden auf Datenoperationen, die unter alleiniger Verwendung des PMA-Interface nicht oder nur umständlich realisiert werden können.[46]

Grundsätzlich empfiehlt es sich, soweit möglich zunächst die Funktionen von PMA zu verwenden. Wenn die Grenzen von PMA erreicht sind, modifiziert man den von PMA erzeugten ↗SQL-Code in der erforderlichen Weise.

Üblicherweise unterscheidet man in ↗SQL zwischen der Data Definition Language (DDL), der Data Query Language (DQL) und der Data Ma-

[46] Als Beispiel dient eine transkribierte und tokenisierte Rede von Angela Merkel, gehalten am 19.4.2010 in Wittenberg aus Anlaß des 450sten Todestages von Philipp Melanchthon (https://www.bundeskanzlerin.de/ContentArchiv/DE/Archiv17/Reden/2010/04/2010-04-19-rede-schlosskirche-wittenberg.html)

nipulition Language (DML). Statements der DDL (wie z.B. create, drop, alter) beziehen sich auf die Datenstrukturen, Statements der DQL dienen der Abfrage bzw. Datenanalyse (z.B. select) und Statements der DML (wie z.B. insert, delete, update) beziehen sich auf die Inhalte der Datenstrukturen.[47]

5.2.2.1 Allgemeine Regeln und Hinweise

- Die Elemente der ↗SQL-Syntax können groß oder klein geschrieben werden:

```
select * from tokens;
```

ist identisch mit

```
SELECT * FROM TOKENS;
```

- ↗Tabellen- und Feldnamen sollten nur dann zwischen sog. „Backticks“ (↗ASCII x60) geschrieben werden, wenn ↗Tabellen- oder Feldnamen aus Sicht von MySQL mißverständlich sind, wie z.B. bei der Bezeichnung „Alter“ (etwa in einer ↗Tabelle mit Informantendaten): ALTER ist ein ↗SQL-Befehl, mit dem Tabellendefinitionen geändert werden können (engl. alter).

```
select `tokens`.`pos` from `tokens`;
```

- Die Adressierung von Feldnamen kann nötigenfalls kaskadierend unter Nennung des Datenbank- und Tabellennamens erfolgen, wobei als Namenstrenner ein Punkt zu setzen ist:

```
datenbankxy.tabellexy.feldxy
```

- Die Paraphrasierung der ↗SQL-Syntax durch Zeilenumbrüche und Einrückungen ist nachdrücklich zu empfehlen.
- Alle ↗SQL-Kommandos sollten mit einem Semikolon (;) abgeschlossen werden.

---

[47] Daneben existieren die DCL (Data Control Language) und die DTL (Data Transaction Language). S. den Überblick unter https://en.wikibooks.org/wiki/MySQL/Language/Definitions: what are DDL, DML and DQL%3F

[48] Der Unterschied wird vielleicht am besten illustriert durch die MySQL-↗Funktion ascii(), die den ↗ASCII-Wert eines Zeichens zurückgibt:

- Zwei doppelte Spiegelstriche gefolgt von einem Spatium (wichtig! Darf nicht fehlen!) leiten einen Kommentar ein, der stets bis zum Ende einer Zeile reicht:

```
select tokens.pos -- Kommentar: Feldauswahl
from tokens;
```

- Zeilenübergreifende Kommentarpassagen werden durch /* ... */ markiert:

```
select
     datei,
     /*
     id,
     zeile,
     pos,
     */
     token
from tokens
where
     token like 'der'
     -- or token like 'die'
     or token = 'das'
;
```

- Die ↗SQL-Syntax legt die Abfolge von Sprachelementen fest. So muss beispielsweise eine where-Klausel unbedingt **vor** einer Gruppierungsanweisung stehen. Eine Abweichung von der definierten Reihenfolge führt zu Fehlermeldungen:

```
select * from tokens where token = 'der' group by pos; -
     - KORREKT
select * from tokens group by pos where token = 'der'; -
     - FALSCH
```

- Folgendes Schema gibt einen Überblick über die festgelegte Reihenfolge bei Verwendung von Select-Statements:

```
SELECT Spaltenname(n)
    FROM Tabellenname(n)
WHERE Bedingung
GROUP BY Spaltennamen(n)
HAVING Bedingung
ORDER BY Spaltenname(n)
LIMIT
;
```

- Für Update-Statements gilt die Reihenfolge:

```
UPDATE Tabellenname
SET Spaltenname1=Wert1, Spaltenname2=Wert2 [, ...]
WHERE Bedingung
;
```

- MySQL kennt einen speziellen Wert „NULL“, der in Datenfeldern verwendet werden kann oder auch als Ergebnis von JOIN-Operationen (s.u. Seite 174 „Join (Tabellenverknüpfung“) auftreten kann. NULL bezeichnet die Abwesenheit jedweder Daten, ist dabei aber zu unterscheiden vom sog. „Leerstring“ (↗String).[48] Als semantischen Unterschied kann man definieren: NULL = „nicht spezifiziert“, Leerstring = „definitiv nicht vorhanden“.

- MySQL-Dokumentation im Internet: http://dev.mysql.com/doc/#manual. Es sollte jeweils die Dokumentation der installierten MySQL-Version verwendet werden. Die Versionsnummer ist abrufbar durch folgenden Befehl:

```
select version();
```

5.2.2.2 Datenselektion

Die Datenselektion (Auswahl von Tupeln/Zeilen/Datensätzen) über die Suchmaske von PMA erlaubt die Eingabe jeweils eines Suchkriteriums pro Tabellenfeld. So lassen sich beispielsweise sämtliche ↗Tokens 'der' selektieren, die in der Spalte POS den Wert 'bestArt' (bestimmter Artikel) enthalten:

[48] Der Unterschied wird vielleicht am besten illustriert durch die MySQL-↗Funktion ascii(), die den ↗ASCII-Wert eines Zeichens zurückgibt:

```
select ascii('');       -- -> 0
select ascii(NULL);     -- -> NULL
```

Für den ↗Leerstring liefert ascii() den ↗ASCII-Wert 0, für NULL wiederum NULL.

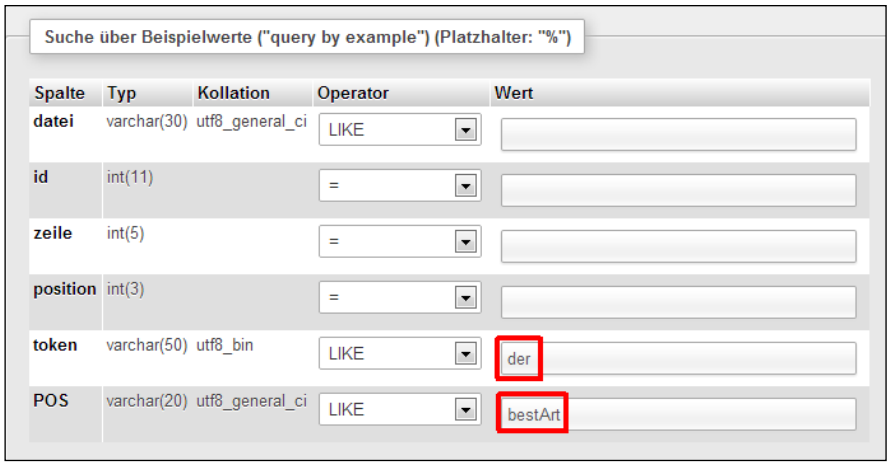


Abbildung 82: Datensuche in PMA mit Kriterienkombination

PMA generiert aus dieser Eingabe die folgende Abfrage:

```
SELECT *
FROM `tokens`
WHERE `token` LIKE 'der'
AND `POS` LIKE 'bestArt'
```

Abbildung 83: Von PMA erzeugtes ↗SQL-Statement

Das Selektionskriterium ist in der WHERE-Klausel formuliert, wobei PMA die beiden über die Suchmaske eingegebenen Bedingungen dahingehend umsetzt, dass beide Kriterien simultan auftreten müssen. PMA formuliert also zwei Bedingungen, die unter Verwendung des Schlüsselwortes AND miteinander verknüpft werden. Als Ergebnis liefert MySQL alle ↗Tokens „der“, die als bestimmter Artikel etikettiert sind. Grundsätzlich wäre jedoch auch die Verwendung von OR möglich, was allerdings ein ganz anderes Suchergebnis zur Folge hätte. Eine gezielte Auswahl eines Operators (AND/OR) über die Suchmaske von PMA ist nicht möglich.

Folgende Abfrage selektiert alle Datensätze mit dem Wert 'der' im Feld ↗Token, die entweder als 'bestArt' oder als 'relPron' etikettiert sind:

```
select *
from tokens
where
     token like 'der'
```

```
  and (
          pos like 'bestArt'
          or pos like 'relPron'
      )
;
```

Die logischen Operatoren AND und OR folgen den Gesetzen der Booleschen Algebra, d.h. AND wirkt inkludierend, weshalb die folgende Abfrage ein leeres Ergebnis liefert:

```
select *
from tokens
where
     pos like 'bestArt'
     and pos like 'relPron'
;
```

Logische Zusammenhänge können durch die Verwendung von runden Klammern eingeflochten werden:

```
select *
from tokens
where
  token like 'd%'
  and (
      pos like 'bestArt'
      or pos like 'relPron'
  )
;
```

Für die Datenselektion stehen mehrere Vergleichsoperatoren zur Verfügung, die sich zum einen hinsichtlich ihrer Verarbeitungsgeschwindigkeit und zum anderen hinsichtlich spezifischer Optionen unterscheiden:

Operator	Bedeutung	Trunkierung
like	ist wie ...	ja
regexp	reguläre Ausdrücke	ja
=	identisch	nein
!=	nicht identisch	nein
>=	größer oder gleich	nein
<=	kleiner oder gleich	nein
between ... and ...	zwischen	nein
in	in definierter Menge	nein
is null	ist „NULL“	nein

Die meisten dieser Operatoren können durch 'NOT' in ihr Gegenteil verkehrt werden. Für die Prüfung, ob ein Feld vollkommen leer ist, **muss** der Operator 'IS NULL' verwendet werden.

Weitere Beispiele:

Verwendung von „between“, „in“ und „is null“ zur ↗Selektion:

```
... where zeile between 5 and 10
```

findet alle Datensätze, in denen im Feld `zeile` ein Wert zwischen 5 und 10 steht (5 und 10 inklusive). „BETWEEN“ wird normalerweise zur Filterung nach Zahlenwerten verwendet.

```
... where token in ('der','die','das')
```

findet alle Datensätze mit einem der gelisteten ↗Tokens im Feld `token`

```
... where POS is null
```

findet alle Datensätze, die im Feld `POS` keinen Wert („NULL“) enthalten.[49]

Die genannten Operatoren können auch im Sinne einer Negation verwendet werden: ... not between ...; ... not in ...; ... is not null ...

Vorsicht ist geboten bei Verwendung des Operators REGEXP. REGEXP (synonym: RLIKE) gestattet die Verwendung von ‚Regulären Ausdrücken‘ (s. S. 81). Dabei kommt eine in MySQL integrierte sog. ‚Engine‘ zum Einsatz, die die Verarbeitung der Regulären Ausdrücke übernimmt. Diese Engine ist **nicht mulitbyte-safe**, was bedeutet, dass bei Verwendung bzw. Vorkommen von ↗UTF-8-kodierten Zeichen unerwartete bzw. falsche Suchergebnisse auftreten können. Zwei Beispiele sollen dies illustrieren:

Sucht man unter Verwendung des Operators REGEXP in einem ↗UTF-8-kodierten Datenbestand nach Wörtern, die aus exakt drei Zeichen/Buchstaben bestehen, so wird das Wort 'für' nicht gefunden:

[49] „is null“ wird unter anderem benötigt, um bei logisch miteinander verknüpften ↗Tabellen Datensätze zu finden, die in der jeweils anderen ↗Tabelle keine Entsprechung besitzen. Vgl. unten (S. 177) im Abschnitt über Tabellenverknüpfungen (sog. Joins).

```
select * from tokens where token regexp '^...$'; --
     findet 'für' nicht
```

Die Erklärung besteht darin, dass ↗UTF-8 den Buchstaben 'ü' mit zwei ↗Byte kodiert, wodurch das Wort 'für' in der Summe aus insgesamt vier ↗Byte besteht. REGEXP geht jedoch davon aus, dass jedes Zeichen mit exakt einem ↗Byte kodiert ist. Entsprechend wird 'für' gefunden, wenn man REGEXP nach Wörtern, bestehend aus vier Zeichen/Buchstaben, suchen läßt:

```
select * from tokens where token regexp '^....$';
-- findet 'für'
```

Ein zweites Beispiel: Der Befehl

```
select * from tokens where token regexp '^Ö';
```

findet ausschließlich ↗Tokens, die mit großem Ö beginnen – was korrekt ist und der Erwartung entspricht. Der Befehl

```
select * from tokens where token regexp '^[Ö]';
```

hingegen – obwohl eigentlich bedeutungsidentisch – findet jedoch auch alle ↗Tokens, die mit ö,Ä, ä, Ü und ü beginnen. Der Grund liegt darin, dass in der ↗UTF-8-Kodierung die Zeichen Ä,Ö,Ü,ä,ö,ü im ersten ↗Byte der Zwei-↗Byte-Kodierung denselben Zahlenwert, nämlich hex c3 aufweisen. Die Regex-Engine von MySQL wertet bei der Interpretation der Zeichenklasse ([Ö]) offenkundig nur das erste ↗Byte aus, was dann zu dem beobachteten Fehler führt.

Beim Einsatz des Operators REGEXP ist also Vorsicht geboten! Es ist zu erwarten, dass dieser Mangel in einer der kommenden MySQL-Versionen behoben sein wird.

5.2.2.3 Projektion

Während sich eine ↗Selektion auf die Auswahl ganz bestimmter Zeilen einer Datenbanktabelle bezieht, ist mit ↗Projektion die Auswahl bestimmter Spalten gemeint. Im Statement „select * from tabelle“ bezeichnet das Sternchen (*) sämtliche Spalten einer ↗Tabelle. Will man nur ganz bestimmte Spalten einer ↗Tabelle angezeigt bekommen, so muss man deren Namen explizit angeben:

```
select spalte1, spalte3, spalte10, spalte4 from tabelle;
```

Das Beispiel zeigt auch, dass man auf diese Weise auch die Reihenfolge der anzuzeigenden Spalten ändern kann. PMA erlaubt die ↗Projektion über den unscheinbaren Link ‚Optionen' links unten auf der Suchmaske (zur Auswahl mehrerer Spalten muss beim Anklicken die Strg-Taste gedrückt werden):

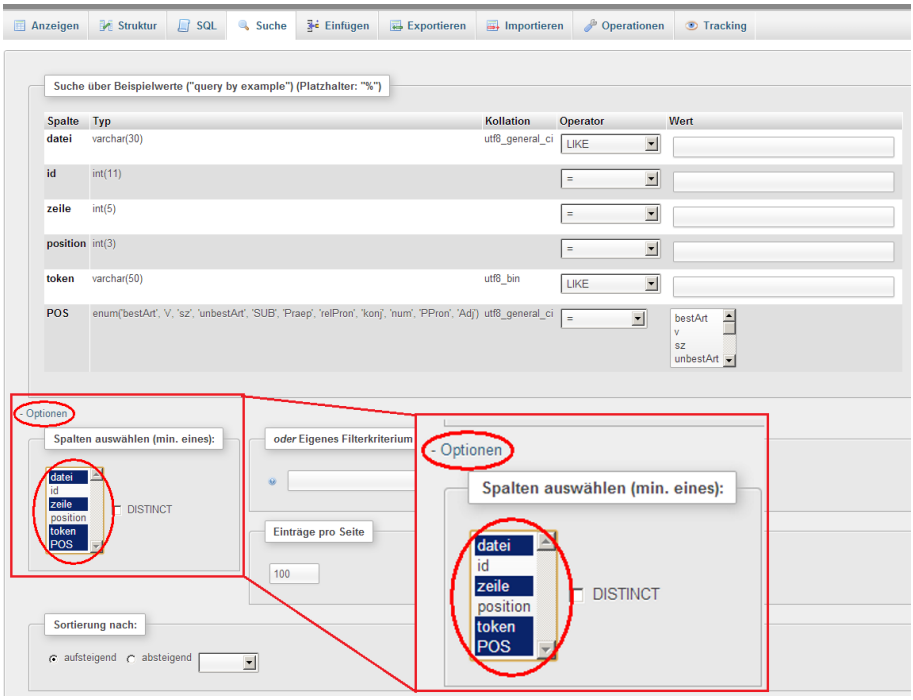


Abbildung 84: Auswahl einzelner anzuzeigender Spalten („↗Projektion")

5.2.2.4 Sortierung

Suchergebnisse können sortiert werden. Dies erfolgt durch das Kommando ORDER BY und die Angabe der Kolumne, nach deren Inhalten sortiert werden soll. Auch die Sortierung nach mehreren Kolumnen ist möglich, in diesem Fall werden die Kolumnennamen durch Kommata getrennt. Beispiel:

```
select *
from tokens
order by datei, zeile asc, position desc
;
```

Das Schlüsselwort DESC (descending) bewirkt absteigende Sortierung, ASC (ascending) das Gegenteil, wobei, wie das Beispiel zeigt, auch ge-

mischte Verwendung möglich ist. Wird weder ASC noch DESC explizit angegeben, erfolgt die Sortierung nach ASC. Im vorliegenden Beispiel wird also das Feld „datei“ aufsteigend sortiert.

5.2.2.5 Gruppierung

Daten können nach Feldinhalten gruppiert werden. Dies erfolgt durch das Kommando GROUP BY

```
select *
from tokens
group byPOS
;
```

GROUP BY bewirkt die Zusammenfassung aller Datensätze mit identischen Einträgen im gruppierten Feld in einem einzigen Datensatz. Dabei wird in den Feldern, deren Inhalt nicht Grundlage der Gruppierung ist, jeweils nur der Inhalt des jeweils ersten Datensatzes angezeigt, die Inhalte der anderen Datensätze unterschlagen. Eine Gruppierung ist eigentlich nur sinnvoll bei gleichzeitiger Verwendung spezieller ↗Funktionen wie z.B. count() und group_concat(). Erstere ermittelt die Anzahl der Datensätze, die durch die Gruppierung zusammengefaßt wurden, letztere konkateniert die Inhalte der Felder, die nicht gruppiert werden:

```
select
   count(*) as anzahl,
   pos,
   group_concat(token separator ' ') as tokens
from tokens
where POS is not null
group by
   POS
order by
   count(*) desc
;
```

Ergebnis:

anzahl	pos	tokens
333	SUB	Familien Gesellschaft Auftrag Geschichte Lebenscha...
197	sz	. . . , . . . , . . . , . . . , . . . , - , . . . , ...
71	bestArt	die Die die der der des Die die der die die Die di...
66	KONJ	und und und auch und Und und und und auch und auch...
56	Praep	für an in für für in In in für in im in in in an i...
32	unbestArt	einem ein eine einen einen eine ein eine eine ein ...
18	PersPron	er es Es es es es Es Er es es es es Er Er er er es...
1	ADV	Sehr
1	relPron	der

Abbildung 85: Suchergebnis mit Gruppierung nach dem Wert im Feld `pos` (Wortart)

5.2.2.6 Datenänderung

Mit dem Kommando UPDATE können vorhandene Datensätze verändert werden:

```
update tokens
   set pos = 'SUB'
where
   token regexp '^[A-ZÄÖÜ]' --         (1)
   and position != 1         --         (2)
;
```

Anmerkungen:
(1) ↗Token beginnt mit Großbuchstaben.
(2) ↗Token steht nicht am Satzanfang.

Sollen gleichzeitig mehrere Felder geändert werden, müssen die entsprechenden Anweisungen durch Kommata getrennt werden:

```
update TABELLE
   set feld1 = 'SUB', feld2 = 'xyz', feld3 = 'abc'
where [Bedingung]
;
```

Es wird dringend empfohlen, zunächst durch select-Statements nur die Datensätze herauszufiltern, in denen die Änderungen vorgenommen werden sollen. Anschließend ändert man das select-Statement in eine update-Anweisung um. Die Verwendung von Kommentarzeichen, die eingefügt oder getilgt werden, ist dabei sehr praktisch:

```
   select * from
-- update tokens
-- set pos = 'SUB'
where
   token regexp '^[A-ZÄÖÜ]'
   and position != 1
;
```

5.2.2.7 Join (Tabellenverknüpfung)

MySQL erlaubt die Verknüpfung zweier oder mehrerer (im Prinzip beliebig vieler) ↗Tabellen (Datenjoin). Zu diesem Zweck wird das Kommando JOIN verwendet.

Gegeben seien die beiden ↗Tabellen:

Mitarbeiter

Name	PLZ
Müller	80539
Meier	90402
Huber	93047

Abbildung 86: ↗Tabelle mit Mitarbeiterdaten

Orte

PLZ	Ort
94034	Passau
80539	München
90402	Nürnberg

Abbildung 87: ↗Tabelle mit Ortsdaten

Eine Verknüpfung dieser beiden ↗Tabellen ohne die Angabe einer Verknüpfungsbedingung kombiniert alle Datensätze der einen ↗Tabelle mit allen Datensätzen der anderen ↗Tabelle:

```
SELECT *
FROM orte
JOIN mitarbeiter;
```

Mathematisch betrachtet, handelt es sich um das kartesische Produkt der Anzahl der Datensätze in den beiden verknüpften ↗Tabellen, also 3*3. Die Ergebnistabelle enthält also neun Datensätze:

PLZ	Ort	Name	PLZ
94034	Passau	Müller	80539
80539	München	Müller	80539
90402	Nürnberg	Müller	80539
94034	Passau	Meier	90402
80539	München	Meier	90402
90402	Nürnberg	Meier	90402
94034	Passau	Huber	93047
80539	München	Huber	93047
90402	Nürnberg	Huber	93047

Abbildung 88: Kombination der beiden ↗Tabellen `Mitarbeiter` und `Orte` durch das Statement JOIN ohne Angabe einer Verknüpfungsbedingung (⇒kartesisches Produkt)

Es ist offenkundig, dass die Kombination der Datensätze nicht sinnvoll ist. Für eine sinnvolle Kombination ist die Angabe einer Verknüpfungsbedingung erforderlich. Diese erfolgt durch die Anweisung ON:

```
SELECT *
FROM orte
JOIN mitarbeiter on (orte.plz=mitarbeiter.plz);
```

Sofern die Felder, deren Inhalt übereinstimmen soll, identische Namen besitzen – so wie hier gegeben – ist eine Art Kurzschreibweise mit der Anweisung USING möglich:

```
SELECT *
FROM orte
JOIN mitarbeiter using (plz);
```

Beide Syntaxvarianten führen zum gleichen Ergebnis. Nun wird den Mitarbeitern der korrekte Ort zugeordnet:

PLZ	Ort	Name
80539	München	Müller
90402	Nürnberg	Meier

Abbildung 89: Kombination der beiden ↗Tabellen `Mitarbeiter` und `Orte` durch das Statement JOIN mit Angabe der Verknüpfungsbedingung, dass die Werte in den Feldern PLZ jeweils dieselben Werte enthalten müssen

Gleichzeitig fällt aber auf, dass die Datensätze der ↗Tabelle „Orte“, die keine Entsprechung in der ↗Tabelle „Mitarbeiter“ besitzen, fehlen. Hier kommt eine Erweiterung der JOIN-Syntax ins Spiel: Mit den Schlüsselwörtern LEFT und RIGHT läßt sich steuern, ob und welche derjenigen Datensätze gezeigt werden, die keine Entsprechung in der jeweils anderen ↗Tabelle aufweisen:

```
SELECT *
FROM orte
LEFT JOIN  mitarbeiter using (plz);
```

liefert als Ergebnis alle Datensätze der linken ↗Tabelle, hier also der erstgenannten ↗Tabelle „Orte“, unabhängig davon, ob diese eine Entsprechung in der rechten ↗Tabelle, hier also der zweitgenannten ↗Tabelle „Mitarbeiter“, haben. Anstelle der nicht vorhandenen Daten erscheint 'NULL', was für 'absolut nichts' steht:

PLZ	Ort	Name
94034	Passau	*NULL*
80539	München	Müller
90402	Nürnberg	Meier

Abbildung 90: Kombination der beiden ↗Tabellen `Mitarbeiter` und `Orte` durch das Statement LEFT JOIN

Vice versa zeigt der Befehl RIGHT JOIN analog alle Datensätze der rechten ↗Tabelle:

```
SELECT *
FROM orte
RIGHT JOIN mitarbeiter using (plz);
```

PLZ	Name	Ort
80539	Müller	München
90402	Meier	Nürnberg
93047	Huber	*NULL*

Abbildung 91: Kombination der beiden ↗Tabellen `Mitarbeiter` und `Orte` durch das Statement RIGHT JOIN

Zum Auffinden von Datensätzen, die in einer der verknüpften ↗Tabellen keine Entsprechung besitzen (z.B. um alle ↗Tokens zu finden, die noch nicht etikettiert wurden), muss/kann man die Bedingung „… WHERE Feld IS NULL“ verwenden:

```
SELECT *
FROM tokens t
LEFT JOIN postags p ON (t.pos=p.tag)
WHERE p.tag IS NULL
;
```

MySQL unterstützt keine direkte Möglichkeit, beide Varianten miteinander zu kombinieren (Fachausdruck: FULL OUTER JOIN). Ein entsprechendes Ergebnis kann – mit Einschränkungen – nur über die Verbindung der beiden Einzelabfrageergebnisse erzielt werden.[50]

„Joins“ können im Rahmen der Korpuslinguistik unter anderem für die Analyse von Datensequenzen wichtig sein. Im folgenden Beispiel wird nach der Abfolge von „bestimmter Artikel“ – „nicht spezifiziertes ↗Token“ – „Substantiv“ gesucht. Zu diesem Zweck wird die ↗Tabelle `Tokens` insgesamt dreimal (mit jeweils anderem Alias- oder Korrelats-Namen) mit sich selbst verknüpft und dabei über die WHERE-Klausel ein entsprechender Filter implementiert:

```
SELECT
    a.zeile AS zeile,
    a.position AS positionA,
    b.position AS positionB,
    c.position AS positionC,
    a.token AS tokenA,
    b.token AS tokenB,
    c.token AS tokenC,
```

[50] Diesem Zweck dient der Befehl UNION ALL (der nur funktioniert, wenn zwei Abfrageergebnisse eine identische Anzahl von Kolumnen liefern).

```
    a.POS AS posA,
    b.POS AS posB,
    c.POS AS posC
FROM tokens a
JOIN tokens b ON ( a.id = b.id -1 )
JOIN tokens c ON ( a.id = c.id -2 )
WHERE
 a.POS =  'bestArt'
 AND c.POS =  'SUB'
```

Diese Abfrage liefert in unserer Musterdatenbank folgendes Ergebnis:

zeile	positionA	positionB	positionC	tokenA	tokenB	tokenC	posA	posB	posC
2	8	9	10	der	größten	Bildungsreformer	bestArt	NULL	SUB
3	16	17	18	der	lutherischen	Reformation	bestArt	NULL	SUB
4	7	8	9	die	berühmte	Tür	bestArt	NULL	SUB
4	14	15	16	die	Martin	Luther	bestArt	SUB	SUB
9	6	7	8	des	christlichen	Glaubens	bestArt	NULL	SUB
13	5	6	7	die	Entdeckung	Amerikas	bestArt	SUB	SUB
16	12	13	14	die	neue	Technik	bestArt	NULL	SUB
18	18	19	20	die	historische	Bedeutung	bestArt	NULL	SUB
24	6	7	8	die	wichtigste	Grundlage	bestArt	NULL	SUB
37	4	5	6	die	damalige	Zeit	bestArt	NULL	SUB
51	3	4	5	die	erforderlichen	Kenntnisse	bestArt	NULL	SUB
62	18	19	20	die	katholische	Kirche	bestArt	NULL	SUB
67	3	4	5	die	Bildungsrepublik	Deutschland	bestArt	SUB	SUB
70	5	6	7	die	richtigen	Voraussetzungen	bestArt	NULL	SUB
73	7	8	9	die	gleichen	Bildungschancen	bestArt	NULL	SUB
75	6	7	8	die	vorschulische	Sprachförderung	bestArt	NULL	SUB
80	9	10	11	die	herkömmlichen	Strukturdebatten	bestArt	NULL	SUB
89	3	4	5	die	Rede	Melanchthons	bestArt	SUB	SUB
91	5	6	7	die	härteste	Arbeit	bestArt	NULL	SUB
92	9	10	11	die	klare	Sprache	bestArt	NULL	SUB
102	1	2	3	Die	große	Mehrzahl	bestArt	NULL	SUB
110	1	2	3	Die	erste	Voraussetzung	bestArt	NULL	SUB

Abbildung 92: Mehrfache Verknüpfung einer Datenbanktabelle mit sich selbst (sog. „self-join“)

Durch die Verknüpfung der ↗Tabelle `token` mit sich selbst über die Bedingung, dass der Wert in den beiden Feldern a.id und b.id um den Wert 1 differieren muss, werden jeweils zwei auf einander folgende Datensätze auf einer Zeile nebeneinander als gleichsam ein neuer Datensatz abgebildet. Dies ermöglicht die ↗Selektion des Aufeinandertreffens von Kriterien in getrennten Datensätzen. Durch die Änderung des Subtra-

henden (-1) läßt sich die Distanz der analysierten ↗Tokens variieren (z.B.: a.id=b.id-2). Durch mehrfachen „self-join“ einer ↗Tabelle läßt sich eine an und für sich beliebige Anzahl von Datensätzen horizontal sequenzieren:

```
SELECT a.token, b.token, c.token
  FROM tokens a
  LEFT JOIN tokens b ON ( a.id = b.id -1 )
  LEFT JOIN tokens c ON ( b.id = c.id -1 )
```

LEFT sorgt hier dafür, dass die Datensätze am Ende, die bei einem „inner join“ nicht auftauchen, da der Maximalwert von id plus Subtrahend nicht existent ist (max(id) + 1 is NULL ⇒ max(id)=NULL-1 — unmöglich/sinnlos!), nicht unterschlagen werden. Zum Nachlesen ist folgende Website zum Thema „joins“ sehr empfehlenswert: http://wiki.selfhtml.org/wiki/Datenbank/Einf%C3%BChrung_in_Joins

5.2.2.8 Funktionen

MySQL besitzt eine große Anzahl von sogenannten „↗Funktionen“. Allgemein gesprochen sind ↗Funktionen kleine Werkzeuge, die bestimmte Aufgaben erfüllen. Grundsätzlich übernehmen ↗Funktionen Daten, formen diese in der einen oder anderen Weise um und geben schließlich die umgeformten Daten als Ergebnis zurück. Daten, die an ↗Funktionen übergeben werden, bezeichnet man auch als „Argumente“. Die Schreibweise eines Funktionsaufrufes folgt stets diesem Muster:

```
Funktionsname(Argument1, Argument2, ArgumentN…).
```

Zwischen dem Namen der ↗Funktion und der öffnenden Klammer darf kein Leerzeichen stehen.

In MySQL kann man im wesentlichen zwei Gruppen von ↗Funktionen unterscheiden: Numerische ↗Funktionen für arithmetische Aufgaben sowie sog. „Stringfunktionen“ für die Verarbeitung von Zeichenketten. Aus Sicht der Korpuslinguistik sind vor allem letztere interessant. Die MySQL-Online-Dokumentation enthält unter der Adresse http://dev.mysql.com/doc/refman/5.6/en/string-functions.html eine übersichtliche Liste aller zur Verfügung stehenden Stringfunktionen samt einer Beschreibung deren Funktionsweise und Aufrufsyntax. Bei der Benutzung der Online-Dokumentation sollte man auf die mit der verwendeten MySQL-Installation korrespondierende MySQL-Versionsnummer achten. Es lohnt sich, die Liste der Stringfunktionen durchzusehen, denn für eine Vielzahl von Aufgaben, die anderweitig nur

schwer oder gar nicht durchzuführen wären, existieren bereits vorgefertigte Lösungen in Gestalt eben dieser Stringfunktionen.

Die MySQL-↗Funktionen werden überdies in zwei weitere Gruppen unterschieden: Sog. Skalarfunktionen, die üblicherweise jeweils auf die Daten eines oder mehrerer Datensätze wirken, und sog. Aggregatfunktionen, die nur im Kontext der Gruppierung von Datensätzen, also bei gleichzeitiger Verwendung einer GROUP BY-Klausel sinvoll sind (s. https://dev.mysql.com/doc/refman/5.6/en/group-by-functions.html#function_group-concat).

Ein Beispiel einer Skalarfunktion wäre z.B. die Ersetzungsfunktion replace(), deren allgemeine Aufrufsyntax folgendermaßen aussieht:

```
REPLACE(str,fromString,toString)
```

Dabei bezeichnet str die Zeichenfolge, innerhalb derer die Ersetzung vorgenommen werden soll. Hierbei kann es sich um ein Literal (geschrieben in Anführungszeichen) oder auch um einen Platzhalter in Form eines Feldnamens handeln. fromString bezeichnet die Zeichenfolge, die innerhalb von str gesucht, und toString die, die dafür eingesetzt werden soll. REPLACE kann bislang leider keine regulären Ausdrücke verarbeiten. Es ist jedoch möglich, mehrere Aufrufe der REPLACE-↗Funktion zu schachteln, wobei das Ergebnis eines REPLACE-Funktionsaufrufes als Argument für den nachfolgenden Aufruf fungiert. Folgendes Statement erzeugt als Ergebnis den ↗String „Hans“:

```
select REPLACE(REPLACE('Haus','u','xx'),'xx','n');
```

Analoge Schachtelungen sind mit *allen* ↗Funktionen möglich, wobei auch die beliebige Mischung unterschiedlicher ↗Funktionen erlaubt ist:

```
select
     char_length(REPLACE(REPLACE('Haus','u','xx'),'xx','
     n'));
```

Ergebnis dieses Statements ist die Zahl 4, nämlich die Länge des ↗Strings, den die Replace-↗Funktionen erzeugt haben, in Buchstaben. Es empfiehlt sich, komplex geschachtelte Funktionsaufrufe durch Zeilenumbrüche und Einrückungen zu strukturieren:

```
select
 char_length(
      REPLACE(
```

```
            REPLACE('Haus','u','xx'),
        'xx','n')
  )
;
```

Aggregatfunktionen wären beispielsweise die ↗Funktion count(), die die Anzahl in einer Gruppe vereinigter Datensätze zählt, oder die ↗Funktion group_concat(), die sämtliche Werte eines oder mehrerer Felder von gruppierten Datensätzen konkateniert:

```
select group_concat(token separator ' ') from tokens
    group by datei, zeile;
```

Dieses Select-Statement gruppiert zunächst alle ↗Tokens, die zur selben ↗Zeile ein und der selben Datei gehören, und gibt dann alle ↗Tokens einer Gruppe als einen linearen ↗String aus. Die Option separator legt dabei fest, welches Trennzeichen zwischen die einzelnen ↗Tokens der Gruppe geschrieben werden sollen. Im vorliegenden Fall ist dies ein Leerzeichen. Wird diese Option weggelassen, setzt MySQL Kommata zwischen die Werte. Für weitere optionale Argumente dieser ↗Funktion sei an dieser Stelle auf die Online-Dokumentation verwiesen.

↗Funktionen können sowohl als Ausgabefelder wie auch im Kontext von WHERE-, GROUP BY-, ORDER BY- und weiteren Klauseln verwendet werden. Auch der Einsatz in update-Befehlen ist möglich.

Es ist stets darauf zu achten, ob eine ↗Funktion „multibytesafe“ ist oder nicht, d.h. ob sie korrekt mit ↗UTF-Kodierungen (z.B. UTF-8) umgehen kann! Ist dies nicht der Fall, erzeugen entsprechende ↗Funktionen unter Umständen ungewollte und sinnlose Ergebnisse.

5.2.3 Nutzung der Kommandozeile

Die Steuerung bzw. Nutzung einer MySQL-↗Datenbank ist nicht nur über PMA, sondern auch über die Kommandozeile einer ↗Shell möglich. Sobald man einigermaßen mit der ↗SQL-Syntax vertraut ist, ist die Nutzung dieser Option sogar in mancher Beziehung PMA vorzuziehen. Die Ergebnisse werden schneller angezeigt und können überdies durch die Verwendung einer ‚↗Pipe‘ direkt an ↗Shell-Kommandos weitergeleitet werden, die dann eine weitere Verarbeitung (Ersetzungen, Umstrukturierungen etc.) übernehmen können.

Der Zugriff auf die ↗Datenbank erfolgt mit einem speziellen Kommandozeilen-Programm, das sowohl für Windows wie auch für

Unix/Linux verfügbar ist. Erfahrungsgemäß ist es schwierig, ausschließlich die sog. Client Binaries, zu denen auch das Kommandozeilen-Programm gehört, von der MySQL-Website herunterzuladen. Vielmehr ist man gezwungen, das sehr umfangreiche MySQL-Komplettpaket downzuloaden, in dem sich auch die Client Binaries befinden. Aus diesem Grund bieten wir die entsprechenden (Windows-/Cygwin-)Binaries (Version 5.5.28; neuere Versionen können von der MySQL-Website heruntergeladen werden[51]) unter folgendem Link zum Download an:

http://www.kit.gwi.uni-muenchen.de/wp-content/uploads/MySQL_ClientBinaries_v5.5.28.zip

Für die Verwendung unter Cygwin müssen die Dateien nach dem Entpacken in das Verzeichnis cygwin\bin kopiert werden. Anschließend ist die Verwendung über die Cygwin-↗Shell möglich. Der Aufruf geschieht gemäß folgender Syntax:[52]

```
mysql -s -h [Serveradresse] -P [Portnummer] -u
[Benutzername] -p[Passwort] -D [Datenbank] --execute
"[SQL-Statement]"
```

Folgende Abbildung zeigt exemplarisch einen mysql-Aufruf, der einen Teil des IPA-Blocks aus einer ↗Datenbank mit der ↗Unicode-↗Codepage selektiert:[53]

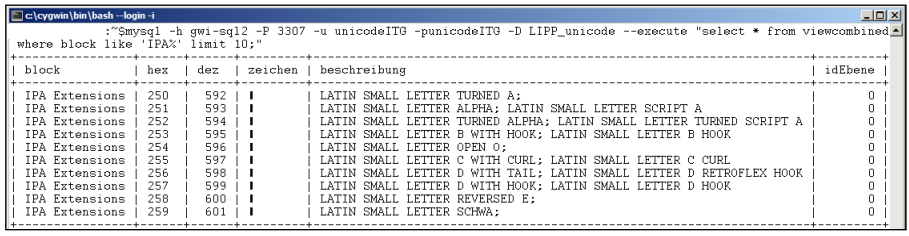


Abbildung 93: Datenbankabfrage mittels Kommandozeilenprogramm von MySQL

Sofern Sie über eine eigene MySQL-Server-Installation verfügen, können Sie sich durch Angabe der entsprechenden Serveradresse jederzeit in

51 https://dev.mysql.com/downloads/

52 Die Passagen in eckigen Klammern müssen – inklusive der eckigen Klammern! – durch die jeweils gültigen Einträge ersetzt werden. Beachten Sie, dass zwischen -p und [Passwort] *kein* Leerzeichen steht!

53 Die Platzhalter in der Kolumne `zeichen` resultieren daraus, dass die Cygwin-↗Shell nicht über geeignete ↗Glyphen verfügt.

der beschriebenen Weise mit Ihrer ↗Datenbank verbinden. Jedoch sind Installation sowie die anschließende Wartung eines MySQL-Servers nicht trivial. Der Betrieb eines MySQL-Servers kann daher an dieser Stelle weder empfohlen noch beschrieben werden.[54]

Der Command-Line-Zugriff auf die MySQL-Server der IT-Gruppe Geisteswissenschaften ist grundsätzlich möglich, aus Sicherheitsgründen wird die entsprechende Berechtigung jedoch nur individuell bei entsprechendem Bedarf an erfahrene Nutzer der MySQL-↗Datenbanken vergeben.

5.2.3.1 MySQL-Batch-Modus

MySQL kann auch im sog. „Batch-Modus" verwendet werden. Das bedeutet, dass man MySQL nicht aufrufen und dann sämtliche Befehle direkt in MySQL eingeben muss, sondern dass man die ↗SQL-Befehle in eine eigene Text-Datei schreiben und MySQL die Befehle dann aus dieser Datei lesen lassen kann. Alle Suchergebnisse werden dann direkt in der Unix-↗Shell ausgegeben. Die entsprechende Syntax sieht folgendermaßen aus:

mysql -u [MySQL-Benutzername] --password=[MySQL-Paßwort] < [Name der Datei mit den Befehlen]

Beispiel:

Die Datei batch.sql enthalte folgende Befehle:

```
use korpus;
select id_token, dateiname, zeile, position, token
from tokens
where token like 'Mü%'
;
```

Um diese Befehle ausführen zu lassen, ruft man folgenden Befehl auf (lmdstud ist das Paßwort der Beispiel-Datenbank):

```
mysql -u root —password=[xyz] < batch.sql
```

[54] In jüngster Zeit werden kleine Homeserver, z.B. von Synology oder QNAP, vermehrt mit Installationen oder Installations-Möglichkeiten von MySQL ausgestattet.

Man kann das Suchergebnis auch direkt in eine Text-Datei schreiben lassen. Dazu erweitert den Kommandoaufruf um einen entsprechenden Umleitungsbefehl („>“):

```
mysql -u root—password=[xyz] < batch.sql > ergebnis.csv
```

Der Inhalt der Datei ergebnis.csv sieht dann folgendermaßen aus:

Abbildung 94: Abfrageergebnis in einer csv-Datei

Die einzelnen Felder des Suchergebnisses sind durch Tabstops voneinander getrennt (farbliche Markierung). Das Suchergebnis kann dann durch copy-paste z.B. auch in ein Microsoft-Word-Dokument eingefügt und dort in eine ↗Tabelle verwandelt werden:

id_token	dateiname	↗Zeile	position	↗Token
6	1.txt	1	6	Mühle

6 Nachwort und Ausblick

Als die moderne Informationstechnologie in den späten 80er und frühen 90er Jahren des vergangenen Jahrhunderts in Gestalt von Personal Computern (PCs) Einzug in die Geisteswissenschaften hielt, wurden die Geräte zunächst überwiegend als Ersatz für die Schreibmaschine verwendet. Die wissenschaftlichen Methoden blieben unverändert, der Einsatz des PCs diente weit überwiegend dem Zweck, die Ergebnisse der wissenschaftlichen Arbeit auf Papier in Form von Seminararbeiten, Artikeln oder Büchern zu drucken und gegebenenfalls zu publizieren. Die elektronischen Daten wurden dabei häufig als eine Art Durchgangsstation empfunden, die nach Abschluß eines Vorhabens bzw. Projektes, d.h. nach erfolgtem Ausdruck auf Papier, als entbehrlich betrachtet wurden; auch wenn man vielleicht noch eine Zeit lang darauf achtete, wenigstens eine Sicherheitskopie der Dateien im Archiv zu bewahren. Nicht selten kam es dann aber vor, dass solche Daten nur wenige Jahre nach ihrer Archivierung nicht mehr verwendbar waren, da sie von den dann aktuellen Programmen oder Programmversionen nicht mehr gelesen werden konnten. Auch wenn man über diese sehr eingeschränkte Wiederverwendbarkeit nicht erfreut war, zumal unter Umständen Passagen, die man gerne wiederverwendet hätte, erneut eingetippt werden mussten, empfand man Dergleichen nicht als eine große Katastrophe, denn schließlich waren die Texte ja schon gedruckt worden.[55]

Dass man die neue Technologie im Sinne einer methodischen Ergänzung auch zur Datenanalyse und somit zum zusätzlichen Erkenntnisgewinn einsetzen kann, war lange Zeit nur Wenigen bewusst. Von denen, die sich dessen bewußt waren, scheuten wiederum nicht wenige davor zurück, sich auf die ungewohnte Arbeitsweise – die Notwendigkeit der logischen Datenstrukturierung, den Umgang mit ↗Datenbanken usw. – einzulassen.

---

[55] Diesem Bedürfnis nach einer im Grunde „analogen“ (als Gegenstück zu „digitalen“) Nutzungsweise der neuen Technologie kamen die Entwickler von ↗Betriebssystemen und Anwendungsprogrammen, allen voran Apple und Microsoft, entgegen, indem sie alles daran setzten, auf dem Computer die Illusion von analoger Wirklichkeit zu erzeugen. Begriffe bzw. Konzepte wie „Desktop“ („Schreibtisch“), WYSIWYG („What you see is what you get“) und „Windows“/ „Fenster“ oder auch die Entwicklung des sog. „Portable Document Format“ (PDF), das als elektronisches Papier wahrgenommen werden soll, sind Ausdruck dieser Bemühungen.

Nach unserer Beobachtung findet im Bereich der Geisteswissenschaften in den letzten Jahren ein grundlegender Umbruch statt. Allenthalben werden nun die Möglichkeiten und Vorteile der Arbeit mit strukturierten Daten und ↗Datenbanken erkannt und genutzt. Waren es anfänglich nur wenige Fächer, die in dieser methodischen Perspektive voranschritten, wie etwa die Sprachwissenschaft und dort speziell die Korpuslinguistik, so weitet sich der Einsatz der entsprechenden Konzepte und Methoden jetzt mehr und mehr auch auf andere geistesiwissenschaftliche Disziplinen wie etwa die Geschichtswissenschaften oder die Archäologie aus.

Wir hoffen und erwarten, dass sich diese positive Entwicklung fortsetzt und diese moderne Arbeitsweise künftig eine noch weitere Verbreitung in den Geisteswissenschaften finden wird. Gerade an den seit einigen Jahren sich auch im Wissenschaftsbetrieb etablierenden Digital Humanities (DH) lässt sich diese erfreuliche Entwicklung hinsichtlich der Verwendung und der Integration von informatischen Methoden in geisteswissenschaftlichen Disziplinen deutlich erkennen. Speziell im Hinblick auf Nachnutzung, Nachhaltigkeit und Langzeitarchivierung, wie sie seit geraumer Zeit verstärkt von Drittmittelgebern verlangt werden, stellt die Verwendung von sauber kodierten und strukturierten Daten aus unserer Sicht einen idealen Lösungsansatz dar. Das größte Potential steckt unseres Erachtens jedoch in der Möglichkeit, den stetig wachsenden Datenbestand projekt- und fachübergreifend miteinander zu verknüpfen und auf diese Weise, unter anderem in einer organisch entstehenden Welt von ↗Metadaten, neue Perspektiven und Fragestellungen zu entwickeln und darauf aufbauend neue Erkenntnisse zu gewinnen. Eine zusätzliche Dynamik kann sich durch den gezielten Einsatz von Internettechnologien wie z.B. den sog. Crowdsourcing-Verfahren entfalten. Diese erlauben eine Kooperation unabhängig von räumlicher Distanz und ermöglichen auch die Einbindung von Laien in die Wissenschaften, wie es bereits von Portalen wie z.B. www.citizen-science-germany.de angestrebt und umgesetzt wird.

7 Anhang

7.1 Checkliste zur Vermeidung unnötiger Probleme

- Fertigen Sie in regelmäßigen Abständen Sicherungen Ihrer Daten an.[56]

- Legen Sie zu Beginn Ihrer Arbeit, speziell vor einer eventuellen Datenerhebung, die Datenstruktur fest und führen Sie, notfalls mit einer geringen Menge fingierter Daten, Tests durch, ob sich damit die von Ihnen verfolgten Ziele und Fragestellungen auch wirklich erreichen bzw. beantworten lassen.

- Der Datenbestand muss durch Vergabe eindeutiger, einheitlicher und konsistenter Dateinamen oder Überschriften gegliedert sein.

- Verwenden Sie bei der Benennung von Dateien und Ordnern grundsätzlich nur ↗ASCII-Zeichen aus dem Bereich zwischen x20 und x7f und nur die, die in der im Anhang gegebenen ↗ASCII-Tabelle in der entsprechenden Kolumne mit „ja“ markiert sind. Neben den dort gelisteten Buchstaben und Zahlen sind ausschließlich das Minuszeichen (-) und der Unterstrich (_) erlaubt. Zur Strukturierung von Dateinamen können Sie auch den Wechsel von Klein- und Großbuchstaben einsetzen (sog. „Camelcase“; z.B.: meineDatenAusKalabrien.doc).

- Kodieren Sie niemals Informationen mit Mitteln der Typographie, also etwa durch Unterstreichung, Kursivierung oder farbliche Markierung.

- Vor der systematischen Weiterverarbeitung jedweder Daten ist auf deren einheitliche Kodierung zu achten, sowohl hinsichtlich der Zeichenkodierung wie auch der Kodierung des Zeilenendes (s. oben das Kapitel 3.2 Zeichenkodierung). Dies gilt besonders bei der Sammlung von Daten aus dem Internet. Welche Kodierung vorliegt, ist abhängig von der Webseite, von der die Daten heruntergeladen werden und daher variabel. Eventuell begegnende "Byte Order Marks" (↗BOM) sind zu entfernen.

[56] Konkrete Tipps hierzu finden Sie in Anmerkung 24 auf S. 60.

- Verwenden Sie im Rahmen der Korpuserstellung ausschließlich den Editor „VIM“, dessen Grundeinstellungen auf alle Fälle die im Anhang (S. 210) gelisteten Einstellungen enthalten muss. Vermeiden Sie auf alle Fälle den Einsatz von Textverarbeitungsprogrammen (z.B. Microsoft Word) oder Windows-Editoren (z.B. notepad), auch, wenn es nur um das kurzzeitige Zwischenspeichern bei der Datensammlung geht.

- Organisieren Sie Ihr Material in möglichst „flachen“ Ordnerhierarchien, d.h. vermeiden Sie die Verteilung von Dateien in ein komplexes, tief verschachteltes System von Ordnern. Nutzen Sie stattdessen die Möglichkeit, Ordnungskriterien in kodierter Form in die Namen der Dateien zu integrieren.

- Ordnen Sie bei Datumsangaben in Datei- und Ordnernamen oder in Datenbankfeldern die Elemente in der Reihenfolge Jahr-Monat-Tag an (also JJJJMMTT, JJMMTT, MMTT). Auf diese Weise entsprechen numerische Sortierungen der chronologischen Abfolge.

- Verwenden Sie konsequent die Datei-Extension zur Angabe der in einer Datei enthaltenen Datenart bzw. -struktur. Abgesehen von den ohnehin quasi standardisierten Extensionen wie z.B. .doc, .docx, .xls, .xlsx, .jpg, .mp3, .wav etc, empfehlen wir folgende Konvention:

.csv	„character separated values“; Daten, die in jeder ↗Zeile durch ↗Separatoren (Empfehlung: Tabstops) in jeweils dieselbe Anzahl von Feldern unterteilt sind (= Tabellengestalt)
.txt	Alle Arten von reinen Textdateien, die nicht dem csv-Schema entsprechen und auch keine Microsoft-Word-Dateien sind
.awk	AWK-Skripts
.sed	SED-Skripts
.sql	↗SQL-Statements (-Syntax)
.TextGrid	Praat-Textgrid-Datei

7.2 Glossar

Die im Folgenden verzeichneten Begriffserklärungen erheben nicht den Anspruch auf universelle Gültigkeit und mögen in Einzelfällen auch um-

stritten sein. Es geht im Wesentlichen darum klarzustellen, was die Autoren dieses Skripts unter den entsprechenden Begriffen verstehen, um Mißverständnissen bei der Lektüre vorzubeugen und den Text allgemein verständlicher zu machen.

absoluter Pfad ↗Pfad

Array Eine ↗Variable mit einfachem oder komplexem Index. Allgemein üblich ist die Schreibweise XYZ[123], wobei XYZ für einen (nahezu) beliebigen Namen der ↗Variablen und 123 für den Index der ↗Variablen stehen. Der Index besteht zumeist aus Natürlichen Zahlen (1,2,3 ...), kann jedoch auch Buchstaben enthalten. In letzterem Fall spricht man von einem „assoziativen“ Array, Arrays mit Zahlen als Index heißen „numerische“ Arrays. Diese sind leichter zu verarbeiten und daher stets vorzuziehen. Die Wertzuweisung erfolgt durch den „Zuweisungsoperator“ „=“.

Beispiel: wort[1]="Hallo", wort[2]="Hans"; satzglied["praedikat"]="kommt nach Hause"

ASCII American Standard Code for Information Interchange. Eine Zeichentabelle („↗Codepage“), entstanden in den 1960er Jahren, die eine Zuordnung der Buchstaben des englischen Alphabets, der arabischen Ziffern und ausgewählter Satz-, Sonder- und Steuerzeichen zu den Natürlichen Zahlen zwischen 0 und 127 vornimmt (kodierbar mit insgesamt 7 ↗Bit). Diese Konvention ist in der Computertechnologie über alle ↗Betriebssystem- und Anwendungsgrenzen hinweg weithin akzeptiert.

Betacode Ursprünglich bezeichnet „Betacode“ eine spezielle i.J. 1970 entwickelte ↗Codepage, die es ermöglicht, den vollständigen Zeichenvorrat der altgriechischen Schrift durch ausschließliche Verwendung von ↗ASCII-Zeichen darzustellen (die im Thesaurus Linguae Graecae versammelte komplette altgriechische Literatur ist bis zum heutigen Tag in dieser Gestalt gespeichert). Darüberhinaus bezeichnen wir jedes entsprechende Verfahren, das der Kodierung „exotischer“ Schriftsysteme durch

ausschließliche Verwendung der ↗ASCII-Zeichen dient, als „Betacode“.

Betriebssystem Engl. operating system („OS“). Betriebssysteme stellen die elementaren Funktionen zur Verwendung eines Computers bereit, wie etwa die Verfahren für Dateneingabe und Datenausgabe, die Verwaltung des ↗Dateisystems inklusive Kopieren und Löschen, die Anwendung von Zugriffsrechten sowie die Verwaltung von Prozessen und Rechnerressourcen. Die gängigsten Betriebssysteme sind Windows und Unix (zu Letzterem ist auch MacOS X zu rechnen).

big endian „big endian“ (‚groß-endig‘) (be) und „little endian“ (‚klein-endig‘) (le) bezeichnen die Reihenfolge, in der bei mehrgliedrigen Zahlen deren Einzelelemente genannt werden. So wird die Zahl 24 im Deutschen nach dem Prinzip „big endian“ (vierundzwanzig ⇒ 4 < 20), im Englischen hingegen nach dem Prinzip des „little endian“ (twenty four ⇒ 20 > 4) bezeichnet. In der Computertechnologie spielen be bzw. le z.B. bei der Zeichenkodierung nach dem ↗UTF-16-Verfahren eine Rolle, bei dem jedes Schriftzeichen durch jeweils zwei ↗Byte repräsentiert wird, wobei das ↗Byte mit dem höheren Zahlenwert auf das mit dem niedrigeren folgen kann (be) oder eben umgekehrt (le).

Bit Ein Bit bezeichnet eine Speicheradresse, die maximal eine „binäre“ Information (ja/nein, schwarz/weiß, 0/1) tragen kann. In der Computertechnologie erfolgt die Realisierung über eine winzige Speicheradresse, an der entweder Stromspannung anliegt oder eben nicht. Jeweils acht Bit zusammen bilden ein ↗Byte.

BOM „Byte Order Mark“. Manche Textverarbeitungsprogramme und Editoren, u.a. Microsoft Office und VIM, schreiben unter Umständen an den Beginn einer multibyte-kodierten Datei standardisierte Zahlenfolgen, die angeben sollen, welches Kodierungsverfahren beim Anlegen der Datei verwendet wurde. Am häufigsten begegnen die Zahlen xEF xBB

xBF (↗UTF-8; Darstellung im Text: ï»¿), xFE xFF (↗UTF-16 ↗big endian; þÿ), xFF xEF (↗UTF-16 little endian; ÿþ). Der BOM ist für die korrekte Darstellung einer Textdatei nicht zwingend erforderlich und wirkt sich bei der automatischen Datenverarbeitung häufig störend aus. Es ist daher zu empfehlen, auf den BOM zu verzichten bzw. ihn ggf. zu entfernen (z.B. mit sed; Beispiel: sed 's/\xEF\xBB\xBF//g'; in VIM: :se nobomb)

Byte — Ein Byte ist stets eine Abfolge von acht Positionen (↗Bit) mit Nullen und/oder Einsen. Für die Kombination von Nullen und Einsen über acht Stellen gibt es insgesamt 256 Möglichkeiten (2*2*2*2*2*2*2*2 = 2^8), was bedeutet, dass sich so die dezimalen Zahlen von 0 bis 255 darstellen lassen. Computersysteme lesen Daten fast ausnahmslos byte-weise. Bisweilen werden dabei jeweils mehrere Byte als ein gemeinsamer Informationsträger zusammengefaßt (↗UTF).

Codepage — Eine wenigstens zweispaltige ↗Tabelle, die eine wechselseitige Zuordnung von Zeichen enthält. Häufig dienen Codepages der Zuordnung von alphanumerischen Einzelzeichen zu Zahlen, damit diese von Computern verarbeitet werden können (A ⇒ 65; z.B. ↗ASCII, ↗Unicode). Auf höherer Ebene können in codpages auch mehrere alphanumerische Einzelzeichen einem anderen alphanumerischen Einzelzeichen zugeordnet werden (z.B. im vom Thesaurus Linguae Graecae verwendeten sog. ↗Betacode: *A)= ⇒ Ἀ)

csv — character separated values – Daten in einer Textdatei in Tabellengestalt, wobei die Gliederung in Kolumnen durch ein eindeutiges Zeichen (character = ↗Separator) erfolgt.

Cursor — In Textverarbeitungsprogrammen, Editoren sowie der ↗Shell die meist blinkende Einfügemarke in Form eines senkrechten Strichs oder eines kleinen Rechtecks, an deren Position bei Betätigung einer Taste auf der Tastatur das entsprechende Zeichen eingefügt wird.

Dateisystem	engl. Filesystem, die Art und Weise, wie ein ↗Betriebssystem Dateien und Ordner auf einem Datenträger ablegt und verwaltet. Neben den Informationen, welche Dateien und Ordner wo zu finden sind, beinhalten moderne Dateisysteme die Möglichkeit der Verwaltung von Zugriffsrechten (welcher User darf in welcher Weise auf welche Dateien und Ordner zugreifen). Beispiele für Dateisysteme sind NTFS (modern), FAT32 (älter; eingeschränkte Zugriffsverwaltung), HFS+ (MacOS X)
Datenbank	Datenbanken sind strukturierte Sammlungen (elektronischer) Daten. Im vorliegenden Skript sind damit ausschließlich sog. „relationale" Datenbanken gemeint, die stets aus einer oder mehreren ↗Tabellen (Fachbegriff: „Relationen") bestehen, in denen die Daten abgelegt sind. Alle ↗Tabellen zusammen bilden ein sog. „Datenbankobjekt". Für die Arbeit mit einer Datenbank ist grundsätzlich ein sog. „Datenbankmanagmentsystem" (DBMS) erforderlich (z.B. MySQL).
Eingabedatei	Datei, mit zu verarbeitendem Primärtext.
Etikettierung	↗Tagging
Feldtrenner	Ein Feldtrenner ist ein eindeutiges, im Grunde frei definierbares Zeichen oder auch eine Zeichenfolge, die eine Zeichenkette in von einander getrennte Felder zerlegt. Die Feldinhalte (↗Token) sind dabei definitionsgemäß stets die Zeichenketten zwischen den Feldtrennern. Eine Textdatei ist üblicherweise durch Feldtrenner unterschiedlicher Ordnung in Felder gegliedert: Die Zeilenenden (kodiert durch die Zahlenwerte 0d 0a [Windows] oder 0a [Unix]) bilden die Feldtrenner erster Ordnung und zerlegen den Text in ↗Zeilen, die Leerzeichen (Zahlenwert 20) sind Feldtrenner zweiter Ordnung und untergliedern die ↗Zeilen in Wörter (jedenfalls zumeist; sofern es sich nicht z.B. um Zahlen oder Gedankenstriche handelt). Eine Textdatei, die in jeder ↗Zeile eine identische Anzahl von Feldtrennern aufweist, ist eine ↗Tabelle.

Filesystem	↗Dateisystem
Fließtext	Ein Fließtext ist die schriftliche Abbildung von (normalerweise) natürlicher Sprache. Die einzelnen ↗Zeilen weisen, im Unterschied zu einer ↗Tabelle, eine variable Anzahl von ↗Feldtrennern (in diesem Fall Leerzeichen) auf.
Funktion	Eine Funktion übernimmt Daten und verarbeitet sie. In den meisten Fällen liefert eine Funktion das Ergebnis ihrer Datenverarbeitung. Dieses Ergebnis nennt man Rückgabewert. Es gibt jedoch auch Funktionen, die beispielsweise die Anzahl von ihr durchgeführter Detailoperationen (etwa Anzahl durchgeführter Ersetzungen; vgl. gsub in AWK) oder auch gar keine Daten zurückliefern (keinen Rückgabewert besitzen; „void“).
Glyphe	Die konkrete Ausformung eines idealisierten Schriftzeichens. Beispiel: Der Buchstabe A (das idealisierte Schriftzeichen) kann in einer nahezu unendlichen Anzahl von Gestalten erscheinen, z.B. kursiv, fett, mit Serifen, ohne, als prächtige Initiale in mittelalterlichen Handschriften etc. Jede einzelne dieser Repräsentationen wäre eine Glyphe.
Homeverzeichnis	Das persönliche Datenverzeichnis eines Computerbenutzers, auf das normalerweise nur er Zugriff hat. Das Homeverzeichnis läßt sich wie folgt ermitteln: 1. Windows-↗Shell (Kommando „cmd“): echo %homedrive%%homepath% 2. Unix-Systeme: echo $HOME oder echo ~ (die Tilde kann stets als Kürzel für das Homeverzeichnis verwendet werden). Der Wechsel ins Homeverzeichnis erfolgt auf Unix-Systemen durch cd ~ oder einfach nur durch cd.
HTML	„Hyper Text Markup Language“, ist eine Art „Sprache“, die ursprünglich unter ausschließlicher Verwendung von ↗ASCII-Zeichen die betriebsystemunabhängige Beschreibung von formatiertem Text inklusive Grafiken erlaubt. In HTML sind die meisten Internetseiten verfaßt. Die herausragende Innovation bestand ursprünglich in den sog. „Hy-

	perlinks“, die auf andere Textstellen, Dokumente und Webseiten verweisen.
HTML-Entities	Spezielle ↗ASCII-Zeichenfolgen, die die Darstellung von Sonderzeichen in ↗HTML-Dokumenten erlauben. Eine HTML-Entity beginnt stets mit einem „Ampersand“ (&) und wird mit einem Strichpunkt abgeschlossen. Beispiel: ä = ä. Durch Kombination mit Zahlenwerten lassen sich beliebige Unicodezeichen darstellen: ʃ = ʃ (hexadezimale Schreibweise), ʃ = ʃ (dezimale Schreibweise).
Kollation	Mit Kollation werden im Kontext der Datenbanktechnologie Algorithmen bezeichnet, die bei der alphabetischen Sortierung sowie beim Vergleich von Zeichenketten angewandt werden. Die in MySQL-↗Datenbanken gängigsten Kollationen sind utf8_unicode_ci, utf8_general_ci und utf8_bin. "_ci" steht dabei für "case insensitive", d.h. die entsprechenden Kollationen berücksichtigen Unterschiede in der Groß-/Kleinschreibung nicht.
Kommando	Anweisung in einer ↗Shell oder in einem Programm, eine bestimmte Tätigkeit auszuführen (z.B. „print“, „echo“, „ls“).
Leerstring	↗String
Lemma	Lexikalische Grundform eines Wortes
little endian	↗big endian
Metadaten	Alle Arten von Daten, die einen Gegenstand (die Primärdaten) beschreiben (synonym: Etiketten, Tags, Annotationen)
Normalisierung	Im Zusammenhang mit relationalen ↗Datenbanken versteht man unter „Normalisierung“ die Strukturierung des Datenbestandes mit dem Ziel, Datenredundanz zu vermeiden (und damit die Gefahr der Entstehung von Dateninkonsistenz zu reduzieren) sowie möglichst effiziente Suchabfragen zu ermöglichen. Grundprinzip ist dabei die Gliederung der Daten in Großeinheiten („Entitäten“) mit spezifischen Eigenschaften. Entitäten werden häufig in

je eigenen ↗Tabellen zusammengefaßt (z.B. ↗Tabellen `Tokens`, `↗Types`, `Informanten` etc.).

Operating System (OS) ↗Betriebssystem

Pfad — Ausgehend von einem bestimmten Punkt im Dateisystem die Angabe der Abfolge von Ordnern/Verzeichnissen bis hin zu dem zu bezeichnenden Objekt (Datei oder Ordner). Startet die Abfolge im Wurzelverzeichnis (↗root) oder sogar mit der Angabe des Rechnernamens, auf dem das Objekt liegt, so spricht man von einem „absoluten Pfad", ansonsten von „relativen" Pfaden.

Pipe, „pipen" — In einer ↗Shell ist es möglich, das Ergebnis eines Kommandos, das normalerweise auf dem Bildschirm ausgegeben wird, direkt an ein anderes Kommando weiterzuleiten, das die Ausgabe des vorangegangenen Kommandos weiterverarbeitet. Auf diese Weise lassen sich im Prinzip beliebig viele Kommandos miteinander verbinden. Als Pipe wird auch das Symbol „|" bezeichnet, das zwischen die einzelnen Kommandos geschrieben werden muss, um die Weitergabe der Daten zu veranlassen.

Primärdaten — Der Kerndatenbestand eines Korpus, inklusive Referenzsystem, ohne ↗Metadaten.

Projektion — Auswahl einzelner Spalten (Attribute) einer (↗Datenbank-)tabelle. S. auch ↗Selektion

Referenzsystem — Alle Daten, die zum Auffinden von Textausschnitten im Gesamtkorpus erforderlich sind (etwa Seitenzahlen, ↗Zeilen-, Intervall- und Positionsnummern, Paragraphen, Abschnittsnummern etc.)

relationale Datenbank ↗Datenbank

relativer Pfad ↗Pfad

root — 1. Auf Unix-Systemen Bezeichnung des Administrators (auch „Superuser"). 2. Bei baumartig verzweigten ↗Dateisystemen spricht man vom „Rootverzeichnis" („Wurzelverzeichnis"), als dessen „Kindelemente" sämtliche auf einem Datenträger

	(Festplatte, CD/DVD, USB-Stick etc.) vorhandenen Dateien und Ordner erscheinen. Auf Unixsystemen dient der Schrägstrich „/„ als Kürzel für das Rootverzeichnis („cd /„ wechselt dorthin).
Rückgabewert	↗Funktion
Selektion	Auswahl einzelner Zeilen (Datensätze) einer (↗Datenbank-)↗Tabelle durch kriterienbasierte Filterung. S. auch ↗Projektion
Separator	↗Feldtrenner
Shell	Von engl. für „Hülle, Mantel, Muschel“, da es sich um einen Datenverarbeitungsprozeß handelt, der sich gleichsam wie eine Muschel um den Betriebssystemkern herum abspielt. Umgangssprachlich wird als Shell meist ein Kommandointerpreter in Fenstergestalt bezeichnet (auch: „Terminal“), in den über die Tastatur Befehle eingegeben werden können, die dann von der Shell ausgeführt werden.
SQL	„Structured Query Language“. Eine (deklarative) Abfragesprache, die in relationalen ↗Datenbanken (z.B. MySQL) verwendet wird.
String	Eine Kette alphanumerischer Zeichen inklusive Satz- und Sonderzeichen, die keine Zahl darstellen soll (Beispiele: Haus, ABC, Super-8). Eine Zeichenkette bestehend aus 0 Zeichen wird als „Leerstring“ bezeichnet.
Tabelle	Eine Tabelle ist ein aus einer oder mehreren ↗Zeilen bestehender Text, dessen ↗Zeilen eine jeweils identische Anzahl von ↗Feldtrennern enthalten.
Tagging	Die Verknüpfung von ↗Primärdaten mit ↗Metadaten (synonym: Etikettierung).
Terminal	↗Shell
Token	Zeichenkette zwischen ↗Separatoren. Sofern nicht eigens anders definiert, gelten sog. „Whitespace-Characters“ (also „Blank“ [= Space, Spatium, Leerschlag] und Tabstop [= Tabulator, Tab]) als ↗Separatoren. Alle Zeichenfolgen dazwischen gel-

ten, unabhängig von ihrer Art und der Anzahl der Zeichen, als Token. Beispiele für Tokens:
Haus[sep]254[sep]53a[sep]xyz[sep]"§%R%Ha
In ↗Fließtexten repräsentiert in den meisten Fällen jedes Wort ein Token, die beiden Bezeichnungen sind in diesem Kontext also überwiegend bedeutungsidentisch.

Type — Ein Type ist eine kontextfreie, idealtypische Repräsentation eines ↗Tokens, die hinsichtlich ihrer morphologischen und grammatischen ↗Etikettierung singulär ist. Beispiel: „<token>Der</token> Herr und <token>der</token> Hund kommen nach Hause“. Der Artikel „der“ taucht in diesem Beispiel zweimal als ↗Token auf. Beide Male handelt es sich um Instanzen des Types „der“, morphologisch/grammatisch etikettiert als „Artikel, maskulin, Nominativ singular“. Das orthographisch identische Relativpronomen „der“ würde bei Einbeziehung der morphologischen und grammatischen Kategorien einen eigenen Type darstellen.

Unicode — Unicode ist eine Zeichentabelle (engl. „↗Codepage“), in der den Schriftzeichen der unterschiedlichsten Schriftsysteme jeweils eindeutig Zahlenwerte zugeordnet sind. Erst die Akzeptanz dieser ↗Tabelle als verbindlicher Standard durch das Gros der Softwareentwickler garantiert Fehlerfreiheit bei Datenverarbeitung und –transfer. Unicode ist *kein* ↗Zeichensatz und **keine** Tastaturbelegung. Vgl. ausführlich S. 51ff..

URL — Uniform Resource Locator („einheitlicher Quellenbezeichner“). Meist ist damit eine sog. Internet-Adresse gemeint, wie z.B. http://www.lmu.de. Ganz allgemein gesprochen, gibt eine URL den Ort an, an dem eine Ressource zu finden ist, und die Art und Weise, wie auf diese zugegriffen werden muss, an. Im gegebenen Beispiel bezeichnet http die Zugriffsmethode (http = hypertext transfer protocol) und www.lmu.de den Ort (= Server). Weitere (fiktive) Beispiele für URLs: ftp://username:

	passwort@ftp.uni-xx.de, smb://fs2/allshares/worddoc.doc
UTF	Abkürzung von „Unicode Transformation Format“. Es handelt sich um eine ganze "Familie" von Kodierungsverfahren, die alle dem Zweck dienen, die Zahlen der ↗Unicode-↗Codepage auf ein Vielfaches eines ↗Bytes abzubilden. Das am weitesten verbreitete Verfahren ist UTF-8, das pro Zeichen eine unterschiedliche, jeweils möglichst geringe, Anzahl von ↗Bytes verwendet. Vgl. https://www.dh-lehre.gwi.uni-muenchen.de/?p=13213 sowie https://de.wikipedia.org/wiki/UTF-8. Siehe auch die Grafik unten S. 202.
UTF-8	↗UTF
Variable	In Computerprogrammen eine Art Container mit eindeutiger Benennung, in dem Werte gespeichert werden können, vergleichbar mit einer Schublade, die wechselnden Inhalt haben kann.
Zeichensatz	Ein Zeichensatz ist eine Sammlung von ↗Glyphen mit jeweils ähnlichen Gestaltungsmerkmalen (z.B. mit oder ohne Serifen; Beispiele: Times New Roman, Arial, Garamond).
Zeile	Das aus dem Buchdruck vertraute Konzept der Textzeile hat in der linearen und ununterbrochenen binären Repräsentation eines Textes keine unmittelbare Entsprechung. Genauso wie Buchstaben und alle anderen Zeichen müssen auch Zeilenwechsel durch Zahlen kodiert werden. Der entsprechende Zahlenwert ist nicht einheitlich festgelegt. Windows-Rechner verwenden die beiden Zahlen x0d und x0a, Unix-Rechner (inkl. Apple) nur x0a. Aus informatischer Sicht fungieren diese Zahlen als ↗Feldtrenner erster Ordnung.

7.3 Software-Liste

		verfügbar für			kostenlos
		Win	MacOS X	Linux	
Audacity	Audiobearbeitung	x	x	x	ja
Cygwin	Unix-Emulation	x	-	-	ja
FineReader	Texterkennung	x	-	-	nein
HeidiSQL	MySQL-Client	x	-	-	ja
MacVIM	Editor	-	x	-	ja
Microsoft Excel	Tabellenkalkulation	x	x	-	nein
MySQL	Datenbankmanagement-system	x	x	x	ja
Praat	Transkriptionspro-gramm	x	x	x	ja
Sequel Pro	MySQL-Client	-	x	-	ja
VIM	Editor	x	x	x	ja

7.4 ASCII-Tabelle

binär	oktal	dezimal	hexadezimal	Abkürzung	Schreibweise I	Schreibweise II	Beschreibung	Verwendung in Datei- und Ordnernamen
00000000	000	0	00	NUL	^@	\0	Null character	nein
00000001	001	1	01	SOH	^A		Start of Header	nein
00000010	002	2	02	STX	^B		Start of Text	nein
00000011	003	3	03	ETX	^C		End of Text	nein
00000100	004	4	04	EOT	^D		End of Transmis-sion	nein
00000101	005	5	05	ENQ	^E		Enquiry	nein
00000110	006	6	06	ACK	^F		Acknowledgment	nein
00000111	007	7	07	BEL	^G	\a	Bell	nein
00001000	010	8	08	BS	^H	\b	Backspace	nein
00001001	011	9	09	HT	^I	\t	Horizontal Tab	nein
00001010	012	10	0A	LF	^J	\n	Line feed	nein
00001011	013	11	0B	VT	^K	\v	Vertical Tab	nein
00001100	014	12	0C	FF	^L	\f	Form feed	nein
00001101	015	13	0D	CR	^M	\r	Carriage return	nein
00001110	016	14	0E	SO	^N		Shift Out	nein
00001111	017	15	0F	SI	^O		Shift In	nein
00010000	020	16	10	DLE	^P		Data Link Escape	nein
00010001	021	17	11	DC1	^Q		Device Control 1	nein

							(oft. XON)	
00010010	022	18	12	DC2	^R		Device Control 2	nein
00010011	023	19	13	DC3	^S		Device Control 3 (oft. XOFF)	nein
00010100	024	20	14	DC4	^T		Device Control 4	nein
00010101	025	21	15	NAK	^U		Negative Acknowledgement	nein
00010110	026	22	16	SYN	^V		Synchronous idle	nein
00010111	027	23	17	ETB	^W		End of Transmis-sion Block	nein
00011000	030	24	18	CAN	^X		Cancel	nein
00011001	031	25	19	EM	^Y		End of Medium	nein
00011010	032	26	1A	SUB	^Z		Substitute	nein
00011011	033	27	1B	ESC	^[	\e	Escape	nein
00011100	034	28	1C	FS	^\		File Separator	nein
00011101	035	29	1D	GS	^]		Group Separator	nein
00011110	036	30	1E	RS	^^		Record Separator	nein
00011111	037	31	1F	US	^_		Unit Separator	nein
00100000	040	32	20				Blank	nein
00100001	041	33	21			!		nein
00100010	042	34	22			"		nein
00100011	043	35	23			#		ja
00100100	044	36	24			$		nein
00100101	045	37	25			%		nein
00100110	046	38	26			&		nein
00100111	047	39	27			'		nein
00101000	050	40	28			(		nein
00101001	051	41	29			)		nein
00101010	052	42	2A			*		nein
00101011	053	43	2B			+		nein
00101100	054	44	2C			,		nein
00101101	055	45	2D			-		ja
00101110	056	46	2E			.		nein
00101111	057	47	2F			/		nein
00110000	060	48	30			0		ja
00110001	061	49	31			1		ja
00110010	062	50	32			2		ja
00110011	063	51	33			3		ja
00110100	064	52	34			4		ja
00110101	065	53	35			5		ja
00110110	066	54	36			6		ja
00110111	067	55	37			7		ja
00111000	070	56	38			8		ja
00111001	071	57	39			9		ja
00111010	072	58	3A			:		nein
00111011	073	59	3B			;		nein
00111100	074	60	3C			<		nein
00111101	075	61	3D			=		nein
00111110	076	62	3E			>		nein

00111111	077	63	3F			?		nein
01000000	100	64	40			@		nein
01000001	101	65	41			A		ja
01000010	102	66	42			B		ja
01000011	103	67	43			C		ja
01000100	104	68	44			D		ja
01000101	105	69	45			E		ja
01000110	106	70	46			F		ja
01000111	107	71	47			G		ja
01001000	110	72	48			H		ja
01001001	111	73	49			I		ja
01001010	112	74	4A			J		ja
01001011	113	75	4B			K		ja
01001100	114	76	4C			L		ja
01001101	115	77	4D			M		ja
01001110	116	78	4E			N		ja
01001111	117	79	4F			O		ja
01010000	120	80	50			P		ja
01010001	121	81	51			Q		ja
01010010	122	82	52			R		ja
01010011	123	83	53			S		ja
01010100	124	84	54			T		ja
01010101	125	85	55			U		ja
01010110	126	86	56			V		ja
01010111	127	87	57			W		ja
01011000	130	88	58			X		ja
01011001	131	89	59			Y		ja
01011010	132	90	5A			Z		ja
01011011	133	91	5B			[		nein
01011100	134	92	5C			\		nein
01011101	135	93	5D			]		nein
01011110	136	94	5E			^		nein
01011111	137	95	5F			_		ja
01100000	140	96	60			`	Backtick	nein
01100001	141	97	61			a		ja
01100010	142	98	62			b		ja
01100011	143	99	63			c		ja
01100100	144	100	64			d		ja
01100101	145	101	65			e		ja
01100110	146	102	66			f		ja
01100111	147	103	67			g		ja
01101000	150	104	68			h		ja
01101001	151	105	69			i		ja
01101010	152	106	6A			j		ja
01101011	153	107	6B			k		ja
01101100	154	108	6C			l		ja
01101101	155	109	6D			m		ja
01101110	156	110	6E			n		ja
01101111	157	111	6F			o		ja

01110000	160	112	70			p		ja
01110001	161	113	71			q		ja
01110010	162	114	72			r		ja
01110011	163	115	73			s		ja
01110100	164	116	74			t		ja
01110101	165	117	75			u		ja
01110110	166	118	76			v		ja
01110111	167	119	77			w		ja
01111000	170	120	78			x		ja
01111001	171	121	79			y		ja
01111010	172	122	7A			z		ja
01111011	173	123	7B			{		nein
01111100	174	124	7C			|		nein
01111101	175	125	7D			}		nein
01111110	176	126	7E			~		nein
01111111	177	127	7F	DEL	^?		Delete	nein

7.5 UTF-8-Kodierung

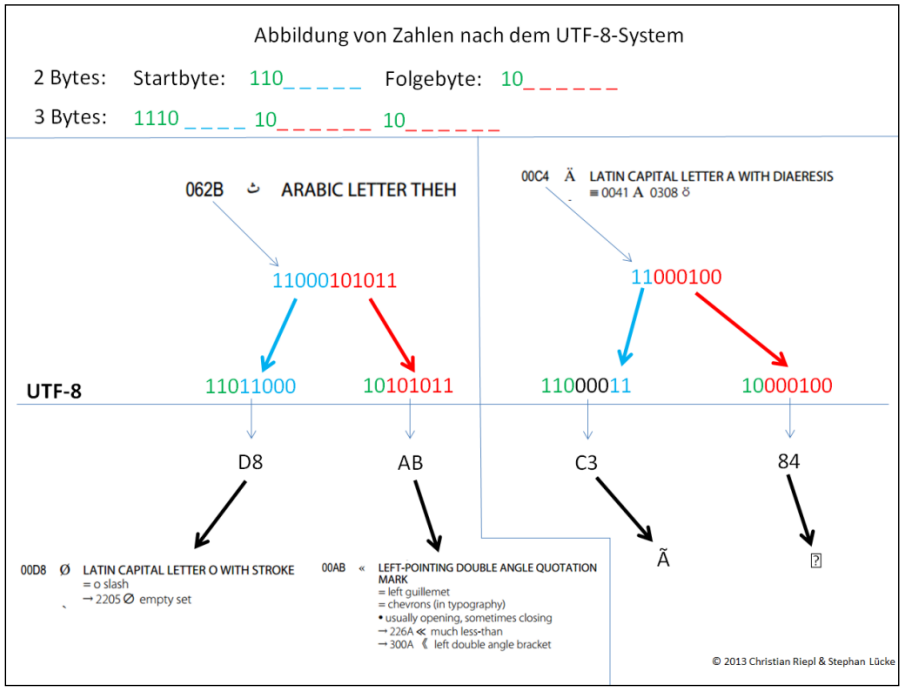


Abbildung 95: Projektion beliebiger binärer Zahlen auf Blöcke zu jeweils acht ↗Bit nach dem ↗UTF-8-Verfahren

7.6 Übersicht über die wichtigsten in AWK eingebauten Variablen

- $0 (gerade gelesene/r Eingabezeile /-record)
- $1 ... $NF (Inhalt der Felder in einer ↗Zeile / einem Record)
- FILENAME (Name der gerade von AWK verarbeiteten Datei)
- FNR (identisch mit NR, nur dass NR bei jeder neu geöffneten Datei, die Zählung in FNR wieder mit 1 beginnt)
- FS (Field ↗Separator)
- NF (Number of Fields; Anzahl der Felder in der gerade gelesenen Eingabezeile)
- NR (Number of Record; Nummer der gerade gelesenen Eingabezeile; bei mehreren ↗Eingabedateien zählt NR die ↗Zeilen über die Grenzen der Dateien hinweg)
- OFS (Output Field ↗Separator)
- RS (Record ↗Separator)

Eine vollständige Liste der in gawk eingebauten ↗Variablen kann unter http://www.gnu.org/software/gawk/manual/gawk.html#Built_002din-Variables abgerufen werden.

7.7 Code-Beispiele und Konfigurationsdateien

Die folgenden Beispiele decken exemplarisch im Bereich der Korpuslinguistik häufig wiederkehrende Anwendungen ab. Die Codes können unter den angegebenen Download-Links heruntergeladen oder mit copy/paste in Textdateien kopiert, diese sodann abgespeichert und anschließend ausgeführt werden.

7.7.1 AWK-Skript: Verwandlung einer (Praat-)TextGrid-Datei in Tabellenstruktur

Das folgende Skript (Download) verwandelt eine vom Programm Praat (s. oben S. 107) erzeugte sog. „Textgrid-Datei“ in eine datenbankgerechte ↗Tabelle. Die Gitterkreuze vor den Zeilen am Anfang des Skripts bedeuten, dass der Text dieser Zeilen als Kommentar aufzufassen ist und keinen ausführbaren Code enthält.

```
# AWK-skript
# Zweck: Verwandlung von (Praat-)TextGrid-Dateien in
     csv-Format
# Datum: 03/2015
# Autor: ITG/slu
# Aufruf: gawk -f tg2csv.awk *.TextGrid
# Ergebnis in Datei tokens.csv
# Klitika müssen innerhalb eines ↗Tokens mit einem =-
     Zeichen (= ↗Separator zweiter Ordnung) abgetrennt
     werden. In diesem Fall darf an anderer Stelle der
     Textgrid-Datei kein =-Zeichen auftreten!
# ACHTUNG: Bei den TextGrid-Dateien auf einheitliche
     Zeichenkodierung (utf-8, latin1 o.ä.) und
     Dateiformat "Unix" achten!

BEGIN {
 OFS = "\t";
 out = "tokens.csv"; # Name der Datei, in die das
     Ergebnis geschrieben wird
 sep1 = " "; # ↗Separator erster Ordnung
 sep2 = "="; # ↗Separator zweiter Ordnung
 printf "id" OFS > out;
 printf "datei" OFS >> out;
 printf "tier" OFS >> out;
 printf "intervall" OFS >> out;
 printf "position" OFS >> out;
 printf "index" OFS >> out;
 printf "xmin" OFS >> out;
 printf "xmax" OFS >> out;
 print "token" >> out;
}

{
 if (FNR==1) {
  print "Bearbeite Datei " FILENAME " ...";
  dateiname=tier=interval=xmin=xmax=token="";
  dateiname=gensub(/\.TextGrid/,"","g",FILENAME);
  on=0;
 }
 if ($0~/item \[1\]:/) {
  on=1;
 }
 if (on==1) {
  if ($0 ~ /^[\t ]*name = /) {
   gsub(/name = /,"",$0);  # call by reference
   tier=gensub(/\"/,"","g",$0); # call by value
  }
```

```
  if ($0~/intervals \[[0-9]+\]:/) {
   match($0,/[0-9]+/,x);
   interval=x[0];
  }
  if ($0 ~ /xmin/) {
   xmin=$NF;
  }
  if ($0 ~ /xmax/) {
   xmax=$NF;
  }
  if ($0 ~ /text = /) {
   gsub(/text = /,"",$0);
   gsub(/## /,"",$0);
   n=split($0,a,sep1);
   for (i=1;i<=n;i++) {
    o=split(a[i],b,sep2);
    for (j=1;j<=o;j++) {
     print ++k,    # id
      dateiname,   # datei
      gensub(/ */,"","g",tier),   # tier
      gensub(/ */,"","g",interval), # intervall
      i,   # position
      j,   # index
      sprintf("%.4f",xmin), # xmin
      sprintf("%.4f",xmax), # xmax
      gensub(/[\",\.\{\}]/,"","g",b[j]) # ↗Token
      >> out;
    }
   }
  }
 }
}

END {
 print "... fertig!";
}
```

Bei Anwendung auf folgende Textgriddatei (Ausschnitt; Download):

```
File type = "ooTextFile"
Object class = "TextGrid"

xmin = 0
xmax = 321.3441950113379
tiers? <exists>
size = 2
item []:
```

```
 item [1]:
     class = "IntervalTier"
     name = "Transkription"
     xmin = 0
     xmax = 321.3441950113379
     intervals: size = 89
     intervals [1]:
         xmin = 0
         xmax = 2.8310381739017227
         text = "Wenkersätze"
     intervals [2]:
         xmin = 2.8310381739017227
         xmax = 4.341183508742201
         text = "eins"
     intervals [3]:
         xmin = 4.341183508742201
         xmax = 9.187696443811646
         text = "Im Winter fliegen die trockenen Blätter
     in der Luft herum"
...
    item [2]:
        class = "IntervalTier"
        name = "ws_nr"
        xmin = 0
        xmax = 321.3441950113379
        intervals: size = 89
        intervals [1]:
            xmin = 0
            xmax = 2.8310381739017227
            text = ""
        intervals [2]:
            xmin = 2.8310381739017227
            xmax = 4.341183508742201
            text = ""
        intervals [3]:
            xmin = 4.341183508742201
            xmax = 9.187696443811646
            text = "1"
...
```

wird folgende ↗Tabelle erzeugt (mit Word zur besseren Darstellung in eine ↗Tabelle verwandelt):

id	datei	tier	intervall	position	index	xmin	xmax	↗Token
1	wenkersaetze	Transkription	1	1	1	0.0000	2.8310	Wenkersätze
2	wenkersaetze	Transkription	2	1	1	2.8310	4.3412	eins
3	wenkersaetze	Transkription	3	1	1	4.3412	9.1877	Im
4	wenkersaetze	Transkription	3	2	1	4.3412	9.1877	Winter
5	wenkersaetze	Transkription	3	3	1	4.3412	9.1877	fliegen
6	wenkersaetze	Transkription	3	4	1	4.3412	9.1877	die
7	wenkersaetze	Transkription	3	5	1	4.3412	9.1877	trockenen
8	wenkersaetze	Transkription	3	6	1	4.3412	9.1877	Blätter
9	wenkersaetze	Transkription	3	7	1	4.3412	9.1877	in
10	wenkersaetze	Transkription	3	8	1	4.3412	9.1877	der
11	wenkersaetze	Transkription	3	9	1	4.3412	9.1877	Luft
12	wenkersaetze	Transkription	3	10	1	4.3412	9.1877	herum
	...							
555	wenkersaetze	ws_nr	1	1	1	0.0000	2.8310	
556	wenkersaetze	ws_nr	2	1	1	2.8310	4.3412	
557	wenkersaetze	ws_nr	3	1	1	4.3412	9.1877	1
558	wenkersaetze	ws_nr	4	1	1	9.1877	10.6627	
559	wenkersaetze	ws_nr	5	1	1	10.6627	15.8956	2
560	wenkersaetze	ws_nr	6	1	1	15.8956	17.3706	
561	wenkersaetze	ws_nr	7	1	1	17.3706	22.7088	3

7.7.2 AWK-Skript zur Verwandlung einer Fließtext-Datei in Tabellenstruktur

```
# AWK-skript
# Zweck: Verwandlung von ↗Fließtext in csv-Format
# Datum: 03/2015
# Autor: ITG/slu
# Aufruf: gawk -f txt2csv.awk *.txt
# Ergebnis in Datei tokens.csv
# Input-Dateien: Pro Zeile je ein Satz
# Klitika müssen innerhalb eines ↗Tokens mit einem =-
    Zeichen (= ↗Separator zweiter Ordnung) abgetrennt
    werden. In diesem Fall darf an anderer Stelle der
    Text-Datei kein =-Zeichen auftreten!
# ACHTUNG: Bei den Input-Dateien auf einheitliche
    Zeichenkodierung (utf-8, latin1 o.ä.) und
    Dateiformat "Unix" achten!

BEGIN {
 OFS = "\t";
 out = "tokens.csv"; # Name der Datei, in die das
    Ergebnis geschrieben wird
 sep1 = " "; # ↗Separator erster Ordnung
```

```
 sep2 = "="; # ↗Separator zweiter Ordnung
 quote = 0;
 printf "id" OFS > out;
 printf "datei" OFS >> out;
 printf "satz" OFS >> out;
 printf "position" OFS >> out;
 printf "index" OFS >> out;
 printf "quote" OFS >> out; # Passagen zwischen
     Anführungszeichen -> 1
 print "token" >> out;
}

{
 if (FNR==1) {
  print "Bearbeite Datei " FILENAME " ...";
  dateiname=satz=token="";
  quote=0;
  dateiname=gensub(/\.txt/,"","g",FILENAME);
 }
 zeile=gensub(/([^ ])(["'.,;:?!\])\}])/,"\\1 \\2
     ","g",$0);
 zeile=gensub(/(["'(\[{])([^ ])/,"\\1 \\2","g",zeile);
 n=split(zeile,a,sep1);
 for (i=1;i<=n;i++) {
  o=split(a[i],b,sep2);
  for (j=1;j<=o;j++) {
   if(b[j]~/["']/ && quote==0) {
    quote=1;
  }
  else if(b[j]~/["']/ && quote==1) {
    quote=0;
  }
   print ++k,  # id
   dateiname,  # datei
   FNR,        # satz
   i,          # position
   j,          # index
   quote,      # quote
   b[j] >> out;# ↗Token
  }
 }
}

END {
 print "... fertig!";
}
```

7.7.3 AWK-Skript zur Veranschaulichung der Aussagenlogik

```
# ITG/slu - 2015-03-23
BEGIN {
 tokens[1] = "Das  ART";
 tokens[2] = "der  ART";
 tokens[3] = "die  ART";
 tokens[4] = "der  ART";
 tokens[5] = "der  ART";
 tokens[6] = "der  ART";
 tokens[7] = "die  ART";
 tokens[8] = "der  PRELS";
 tokens[9] = "die  ART";
 tokens[10] = "Der ART";
 tokens[11] = "der ART";
 tokens[12] = "die ART";
 tokens[13] = "die ART";
 tokens[14] = "das PDS";
 tokens[15] = "die ART";
 tokens[16] = "der ART";
 tokens[17] = "das ART";
 tokens[18] = "die PRELS";
 tokens[19] = "der ART";
 tokens[20] = "das ART";
 tokens[21] = "die ART";
 tokens[22] = "der ART";

 print;
  for (i in tokens) {
       print i, tokens[i];
     }
     print "Start Demo:";
  for (i in tokens) {
    split(tokens[i],a,/[\t ]+/);
    if (a[1]=="der" && a[2]=="ART") { # Konjunktion
     print "Konjunktion! " i, a[1] " : " a[2];
    }
  }
     print;
     for (i in tokens) {
    split(tokens[i],a,/[\t ]+/);
    if (a[1]=="der" || a[2]=="ART") { # Adjunktion
     (Nichtausschließendes Oder)
     print "Adjunktion! " i, a[1] " : " a[2];
     }
     }
     print;
```

```
  for (i in tokens) {
    split(tokens[i],a,/[\t ]+/);
    if ((a[1]=="der" || a[2]=="ART") && !(a[1] == "der"
     && a[2] == "ART")) {   # Disjunktion
     (Ausschließendes Oder)
     print "Disjunktion! " i, a[1] " : " a[2];
     }
  }
     print;
}
```

7.7.4 VIM-Konfigurationsdatei _vimrc

Die Datei _vimrc (Download) (unter Unix .vimrc; Download) muss unbedingt die folgenden Einstellungen enthalten. Sie können diese Einstellungen in eine Textdatei kopieren und diese unter dem Namen „_vimrc“ in Ihrem ↗Homeverzeichnis abspeichern. Ein doppeltes Anführungszeichen am Beginn einer Zeile fungiert als Kommentarzeichen. Auf diese Weise lassen sich temporär unerwünschte Einstellungen deaktivieren (s. letzte Zeile „ " autocmd ...“).

```
" VIM-Konfigurationsdatei zur Verwendung auf Windows-
     Rechner. Abzuspeichern im Homeverzeichnis des Users
     (z.B. c:\users\[Benutzername] oder
     c:\Benutzer\[Benutzername]
source $VIMRUNTIME/mswin.vim " Laden der VIM-Skript-
     Datei mswin.vim, die VIM so konfiguriert, dass es
     sich in wie von Windows-Programmen gewohnter Weise
     bedienen läßt (v.a. spezielle Tastenkombinationen
     wie z.B. ctrl-c -> copy)
behave mswin " Windows-typisches Verhalten, d.h. von
     Windows-Usern gewohnte Tastenkombinationen
     funktionieren auch in VIM (z.B. ctrl-c -> copy und
     ctrl-v -> paste)
set helplang=de " Sprache der Hilfefunktion. "de" meist
     wirkungslos, da nur Englisch verfügbar
set history=200 " Anzahl der in der Datei _viminfo
     protokollierten Befehle und Suchmuster
set hlsearch " Bei Suche nach Zeichenfolgen werden
     Treffer farblich hervorgehoben
set incsearch " Bei Suche nach Zeichenfolgen wird
     jeweils der erste Treffer angezeigt, der dem bis
     dahin eingegebenen Suchmuster entspricht
set keymodel=startsel,stopsel "
set ruler " Anzeige von Zeilen- und Spaltennummer
     (rechts unten)
set selection=exclusive
```

```
set selectmode=mouse,key " Textmarkierung/-auswahl mit
     Maus und Tastatur
set whichwrap=b,s,<,>,[,]
set ignorecase " Groß-/Kleinschreibung bei Suchvorgängen
     irrelevant
" set noignorecase " Groß-/Kleinschreibung bei
     Suchvorgängen relevant
syntax enable " Farbliche Paraphrasierung von Code-
     Syntax
" colorscheme desert " Farbschema mit schwarzem
     Hintergrund
colorscheme default " Standard-Farbschema
set fileformats=dos,unix,mac " Zeilenende-Kodierungen,
     die VIM beim Öffnen einer Datei berücksichtigt bzw.
     auf die eine Datei geprüft wird
set fencs=utf-8,ucs-bom,latin1 " Zeichenkodierungen, auf
     die eine Datei beim Öffnen nacheinander getestet
     wird (in der gegebenen Reihenfolge)
set encoding=utf-8 " Von VIM intern (z.B. beim Schreiben
     in die Datei _viminfo) verwendete Standardkodierung
set fileencoding=utf-8 " Standardkodierung beim
     Speichern von Dateien
set nu " Anzeige von Zeilennumerierung
set tabstop=2 " Anzahl von Bildschirmspalten pro Tabstop
set guifont=Courier New:h12:cANSI " Auswahl des Fonts,
     der von gVim verwendet werden soll. Wichtig ist,
     dass dieser auf dem System auch verfügbar ist
set autoindent " Automatische Übernahme der
     Zeileneinrückung bei Einfügen eines Zeilenumbruchs
" autocmd GUIEnter * simalt ~x " veranlasst gvim, mit
     maximiertem Fenster zu starten
```

7.7.5 Beispiele für Ersetzungskommandos im Editor VIM

```
:%s/<.*>//g
:%s/^\s*\n//g
:%s/ *$//g
:%s/\*[0-9][0-9]*\*//g
:%s/\([0-9][0-9]*\)/\r\1/g
:%s/\n\([a-zA-Z]\)/ \1/g
:%s/\([0-9]\)\n /\1  /g
```

Korpus im Text

Der Band 1 der Schriftenreihe „Korpus im Text“ (KiT) richtet sich hauptsächlich an Studierende der Korpuslinguistik. Das Werk versteht sich als Handreichung für die praktische Arbeit und ist darauf ausgerichtet, in kompakter und allgemein verständlicher Form sowohl die Grundlagen der elektronischen Datenverarbeitung wie auch den Umgang mit spezifischen Werkzeugen (Programmen) zu vermitteln. Es enthält überdies allgemeine Empfehlungen für das methodische Vorgehen. Die grundlagenorientierte Darstellung des Umgangs mit Editoren, Datenbanken und essentiellen Funktionen und Konzepten der Computertechnologie kann auch für Adressaten außerhalb der genannten Zielgruppe von Interesse sein.