

Korpusbasierte Kollokationssuche  
Abschlussbericht  
Stand: 31. Januar 2002

# Inhaltsverzeichnis

<b>I</b>	<b>Einleitung</b>	<b>2</b>
<b>1</b>	<b>Überblick</b>	<b>4</b>
1.1	KoKS: Ziele und Ideen . . . . .	4
1.1.1	Die Motivation hinter KoKS . . . . .	4
1.1.2	Ziele von KoKS . . . . .	5
1.1.3	Anwendungen . . . . .	5
1.2	Architektur . . . . .	5
<b>2</b>	<b>Kollokationen</b>	<b>7</b>
2.1	Von Wortkombinationen zu Kollokationen . . . . .	7
2.2	Kollokationen in der Sprachwissenschaft . . . . .	9
2.2.1	Kollokationen bei Firth . . . . .	9
2.2.2	Automatische Gewinnung von Kollokationen mit statistischen Verfahren . . . . .	9
2.2.3	Kollokationen in Lexikographie und Fremdsprachdidaktik . . . . .	10
2.3	KoKS und Kollokationen . . . . .	12
<b>II</b>	<b>Linguistische Verarbeitung und Datenhaltung</b>	<b>13</b>
<b>3</b>	<b>Korpora</b>	<b>15</b>
3.1	Parallele Korpora . . . . .	16
3.2	Vorhandene Korpora . . . . .	16
3.2.1	Überblick . . . . .	16
3.2.2	Auswahl . . . . .	18
3.3	Quantitative Angaben . . . . .	18
3.3.1	Die Korpora in Zahlen . . . . .	18
3.3.2	Programme zur Auswertung . . . . .	19
3.4	Aufbereitung . . . . .	20
3.4.1	Allgemeines Vorgehen . . . . .	20
3.4.2	Spezielle Probleme . . . . .	21
3.5	Sprachidentifikation . . . . .	23
3.5.1	Mathematischer Hintergrund . . . . .	23
3.5.2	Beispiel . . . . .	24
3.6	Verwaltung und Organisation . . . . .	24
3.6.1	XML . . . . .	24
3.6.2	Korpus-Browser . . . . .	25
3.7	Ausblick und Verbesserungen . . . . .	27
<b>4</b>	<b>Wörterbücher</b>	<b>29</b>
4.1	Einleitung/Motivation . . . . .	30
4.2	Dictionary Entry Parser . . . . .	30
4.3	Aufbau der Textformate . . . . .	31
4.3.1	Einfaches Format . . . . .	31
4.3.2	Das LQL Wörterbuch . . . . .	32

4.4	Zusammenfassung . . . . .	34
4.5	Ausblick . . . . .	35
<b>5</b>	<b>Normalisierung</b>	<b>36</b>
5.1	Motivation . . . . .	37
5.2	Unser Vorgehen . . . . .	37
5.3	Ausgabeformat . . . . .	37
5.4	Filter . . . . .	40
5.5	Tools . . . . .	40
5.6	Ausblick . . . . .	40
<b>6</b>	<b>Tagging</b>	<b>41</b>
6.1	Motivation . . . . .	42
6.1.1	Grundlegendes . . . . .	42
6.1.2	KoKS und Tagging . . . . .	42
6.2	Die Tagging-Programme . . . . .	43
6.3	Ein Beispiel . . . . .	43
6.4	Bewertung und Ausblick . . . . .	46
<b>7</b>	<b>Alignment</b>	<b>47</b>
7.1	Motivation . . . . .	48
7.2	Alignment-Grundlagen . . . . .	48
7.2.1	Übersetzungsrelationen explizieren — Alignment . . . . .	48
7.2.2	Alignment und Korrespondenz . . . . .	53
7.2.3	Alignmentebenen . . . . .	53
7.2.4	Maschinelle Verfahren . . . . .	53
7.2.5	Abstand . . . . .	53
7.2.6	Abstandsmatrix und Alignment-Pfad . . . . .	54
7.2.7	Kontinuum von Abstandsmaß-Methoden . . . . .	54
7.3	Alignment in KoKS . . . . .	55
7.3.1	Abstandsmaße . . . . .	55
7.3.2	Abstandsmatrix . . . . .	68
7.4	Alignment-Tools des KoKS-Projektes . . . . .	70
7.4.1	Kommandozeile . . . . .	70
7.4.2	Visualisierungstool “Mavis” . . . . .	72
7.5	Ausblick . . . . .	74
7.5.1	Zusammenfassung . . . . .	74
7.5.2	Kritik am Konzept . . . . .	74
7.5.3	Absatzalignment . . . . .	74
7.5.4	Evaluation . . . . .	75
<b>8</b>	<b>Phrasenkorrespondenz</b>	<b>76</b>
8.1	Einleitung . . . . .	77
8.1.1	Syntaktische Strukturen erkennen . . . . .	77
8.1.2	Der Ansatz von KoKS . . . . .	77
8.2	Chunk-Parsing mit Tagfolgen . . . . .	77
8.2.1	Tagfolgen für Phrasen des Deutschen . . . . .	78
8.2.2	Tagfolgen für Phrasen des Englischen . . . . .	78
8.3	Die Phrasenzuordnung im Einzelnen . . . . .	80
8.3.1	Tools . . . . .	81
8.4	Ergebnisse . . . . .	81
8.5	Ausblick . . . . .	82

<b>9 Datenbank</b>	<b>84</b>
9.1 Motivation . . . . .	85
9.2 Konzepte . . . . .	85
9.3 Unser Vorgehen . . . . .	85
9.4 Design der KoKS-Datenbank . . . . .	85
9.4.1 Tabellen . . . . .	85
9.4.2 Beschreibung des Designs . . . . .	87
9.5 Tools . . . . .	87
9.6 Ausblick . . . . .	87
<b>III Anwendung</b>	<b>88</b>
<b>10 Beispielapplikation</b>	<b>90</b>
10.1 Motivation . . . . .	91
10.2 Technik und Benutzung (Anwendungsszenario) . . . . .	91
10.3 Middleware . . . . .	93
10.4 Ausblick . . . . .	94
10.5 Resume . . . . .	94
<b>11 Call Kontext</b>	<b>95</b>
11.1 Motivation . . . . .	95
11.1.1 Was ist CALL? . . . . .	95
11.1.2 Konzepte und Anwendungsbereiche . . . . .	95
11.2 Das KoKS-System und CALL . . . . .	95
11.3 Ausblick . . . . .	96
<b>IV Abschließendes</b>	<b>97</b>
<b>12 Projektverlauf</b>	<b>99</b>
12.1 Projektleitersicht . . . . .	99
12.1.1 Übersicht . . . . .	99
12.1.2 Patrick: 18.10.00 - 29.11.00 . . . . .	99
12.1.3 Arno: 6.12.00 - 10.1.01 . . . . .	100
12.1.4 Philip: 19.1.01 - 13.3.01 . . . . .	100
12.1.5 Britta: 20.3.01 - 04.4.01 . . . . .	100
12.1.6 Joachim: 11.4.01 - 30.5.01 . . . . .	101
12.1.7 Norman: 07.06.01 - 11.07.01 . . . . .	101
12.1.8 Arno: 18.7.01 - 4.9.01 . . . . .	101
12.1.9 Philip: 13.9.01 - 24.10.01 . . . . .	101
12.1.10 Norman: 31.10.01 - 26.11.01 . . . . .	101
12.2 Teilnehmer- / Paketsicht . . . . .	101
12.2.1 Philip: Tagging und mehr . . . . .	101
12.2.2 Patrick: Alignment . . . . .	104
12.2.3 Britta: Datenbank und ... . . . .	105
12.2.4 Arno: Ohne Daten keine Ergebnisse . . . . .	106
12.2.5 Norman . . . . .	107
12.2.6 Joachim: Alignment und Anfrageserver . . . . .	107
<b>13 Sitzungsprotokolle</b>	<b>109</b>

<b>V</b>	<b>Dokumentation</b>	<b>192</b>
<b>A</b>	<b>Systemadministration</b>	<b>194</b>
A.1	Software-Installation	194
A.1.1	Überblick	194
A.1.2	Programmiersprachen	194
A.1.3	Systemdienste	195
A.1.4	IMS-Tree-Tagger	195
A.1.5	Datenbank	195
A.1.6	KoKS-Software	196
A.1.7	Konfiguration	196
A.1.8	Funktionstest	196
A.2	System-Administration	197
A.3	Linux Setup	198
<b>B</b>	<b>Meta-Dokumentation</b>	<b>201</b>
B.1	Dokumentation	201
B.1.1	Makrodefinitionen	201
B.2	Präsentationen	201
B.2.1	Präsentation am 13.12.2000	201
B.2.2	Präsentation am 24.06.2001	202
B.2.3	Präsentation am 29.08.2001	202
<b>C</b>	<b>Korpora – Dokumentation</b>	<b>203</b>
C.1	Tools und Optionen	203
C.1.1	Korpus-Verwaltung	203
C.1.2	Korpus-Auswertung	203
C.1.3	XML Tools	204
C.1.4	Sprachklassifikation	208
C.2	Dateiformate	210
C.2.1	DTD	210
C.2.2	Sprachklassifikation	212
<b>D</b>	<b>Normalisierung – Dokumentation</b>	<b>214</b>
D.1	Tools und Optionen	214
D.1.1	normalize.py	214
D.1.2	preproc.py	214
D.1.3	korpus2db.py	214
D.1.4	py2pl.py	215
<b>E</b>	<b>Tagging – Dokumentation</b>	<b>216</b>
E.1	Die Tools im Einzelnen	216
E.1.1	Tokenisierung: tagger/bin/separate-punctuation	216
E.1.2	Tokenisierungskorrektur: bin/metatag-correction	217
E.1.3	Sonderzeichenrekonstruktion: bin/umlaut.pl	217
E.1.4	Tagging und Lemmatisierung: tagger/bin/tree-tagger	218
E.1.5	Satzendenerkennung: punkt-tagger-(german english)	218
E.2	Ausblick: Verbesserungsmöglichkeiten	219
E.3	Die Tagsets	219
E.3.1	STTS (deutsch)	219
E.3.2	Penn-Treebank-Tagset (englisch)	220
E.3.3	Eigene Tags und Annotationen	221
E.3.4	LQL-Tagset (deutsch & englisch)	222

<b>F</b>	<b>Alignment – Dokumentation</b>	<b>224</b>
F.1	Überblick . . . . .	224
F.2	Tools und Optionen . . . . .	224
F.2.1	Abstandsmass . . . . .	224
F.2.2	Matrixausgabe . . . . .	226
F.2.3	Matrixalignment . . . . .	228
F.2.4	Erzeugen alignter Dateien . . . . .	229
F.2.5	Dokumentaushgabe . . . . .	229
F.2.6	Aligner von Church und Gale . . . . .	230
F.3	Dateiformate . . . . .	230
F.3.1	Dokument . . . . .	231
F.3.2	Abstandsmatrix . . . . .	231
F.3.3	Alignment . . . . .	232
F.3.4	Abstandinfo . . . . .	232
F.4	Schnittstellen . . . . .	233
F.4.1	Dokument . . . . .	233
F.4.2	Abstandsmaß . . . . .	233
F.4.3	Alignmentoptimierer . . . . .	233
F.4.4	Ausblick . . . . .	233
F.5	Konzeption . . . . .	234
F.5.1	Dokument . . . . .	234
F.5.2	wichtige Java-Klassen des Aligners (für mavis.jar, al.jar, headingMapper.jar)	235
F.5.3	Das Paket de.denkselbst.matrixAligner . . . . .	236
F.5.4	Das Paket de.denkselbst.matrix . . . . .	241
F.5.5	Das Paket de.denkselbst.mavis . . . . .	250
F.5.6	Das Paket de.denkselbst.kommandozeile . . . . .	251
<b>G</b>	<b>Phrasenalignment – Dokumentation</b>	<b>253</b>
G.1	Tools und Optionen . . . . .	253
G.1.1	getsegs.py . . . . .	253
G.1.2	zuordnung2.py . . . . .	253
G.1.3	phrasealign.py . . . . .	253
G.2	Dateiformate . . . . .	253
G.3	Schnittstellen . . . . .	255
<b>H</b>	<b>Datenbank – Dokumentation</b>	<b>256</b>
H.1	SQL-Skripte zum Erstellen der Datenbank . . . . .	256
H.2	Python-Datenbank-Schnittstelle . . . . .	256
H.2.1	RegionVonDB . . . . .	256
H.2.2	Wörterbücher . . . . .	256
H.3	Abfrage-Schnittstelle . . . . .	257
H.3.1	querydb.py . . . . .	257
H.3.2	Beschreibung der Klassen . . . . .	259
H.4	Korpora und die Datenbank . . . . .	261
H.4.1	Eintragen der Korpora in die Datenbank . . . . .	261
H.4.2	Löschen von Korpora . . . . .	261
<b>VI</b>	<b>Source-Code</b>	<b>262</b>
<b>I</b>	<b>Korpora – Code</b>	<b>264</b>
I.1	Korpus-Verwaltung . . . . .	264
I.1.1	koksUtil.tcl . . . . .	264
I.1.2	denews-add.tcl . . . . .	264
I.2	Korpus-Browser . . . . .	266
I.3	XML Tools . . . . .	284

I.3.1	mapping.dtd	284
I.3.2	meta.dtd	285
I.3.3	idx-ls.tcl	285
I.3.4	idx-add.tcl	289
I.3.5	idx-repair.tcl	292
I.3.6	meta-ls.rb	293
I.4	Sprachklassifikation	294
I.4.1	ngram.rb	294
I.4.2	classifier.rb	296
I.4.3	cfile.rb	298
I.4.4	cserver.rb	300
I.4.5	cclient.rb	301
<b>J</b>	<b>Wörterbuch-Code</b>	<b>303</b>
J.1	Code für die Dictionary-Entry-Parser	303
J.1.1	Beispielcode für einfache Wörterbücher: vbsource.bas	303
J.1.2	Code für das LQL-Wörterbuch	307
<b>K</b>	<b>Normalisierung – Code</b>	<b>323</b>
K.1	normalize.py	323
K.2	preproc.py	327
K.3	py2pl.py	332
<b>L</b>	<b>Tagging – Code</b>	<b>334</b>
L.1	tree-tagger-Skripte	334
L.1.1	tree-tagger-english	334
L.1.2	tree-tagger-german	335
L.2	Sonderzeichenrekonstruktion	335
L.3	Korrektur der Metatags	338
L.4	Satzendenerkennung	338
L.4.1	punkt-tagger-english	338
L.4.2	punkt-tagger-german	341
<b>M</b>	<b>Alignment – Code</b>	<b>347</b>
M.1	Region Paket	347
M.1.1	__init__.py	347
M.1.2	absatz.py	347
M.1.3	alignment.py	349
M.1.4	atom.py	349
M.1.5	dokument.py	350
M.1.6	korpus.py	353
M.1.7	region.py	355
M.1.8	satz.py	359
M.1.9	segment.py	360
M.2	Church and Gale Aligner	361
M.2.1	align.c	361
M.2.2	getopt.c und getopt.h	372
M.3	Weitere Python Module und Skripte	372
M.3.1	abstand.py	372
M.3.2	align.sh	385
M.3.3	align2al.py	386
M.3.4	align_1zu1.sh	388
M.3.5	align_ai.sh	389
M.3.6	align_debug.sh	390
M.3.7	align_mtr.sh	391
M.3.8	align_pur.sh	392

M.3.9	alignment.py	392
M.3.10	alvis.py	393
M.3.11	demo-a2.py	394
M.3.12	demo-abstand.py	396
M.3.13	demo-dok.py	398
M.3.14	dokvis.py	398
M.3.15	eingabe.py	399
M.3.16	getalign.py	405
M.3.17	matrix.py	406
M.3.18	matrix.sh	408
M.3.19	mavis.sh	408
M.3.20	mergemtr.py	408
M.3.21	mtr2mavis.py	411
M.3.22	nmatrix.py	412
M.3.23	zeitmessung.py	417
M.4	dds-Paket	418
M.4.1	Das Makefile align/makefile	418
M.5	Java	419
M.5.1	de.denkselbst.matrix.Matrix.java	419
M.5.2	de.denkselbst.matrix.Matrix.Pfad.java	432
M.5.3	de.denkselbst.matrix.Matrix.KoksZuordnung.java	437
M.5.4	de.denkselbst.matrix.Matrix.MatrixView.java	442
M.5.5	de.denkselbst.matrix.Matrix.MatrixHistogramView.java	455
M.5.6	de.denkselbst.matrix.Matrix.MatrixStatInfoDialog.java	458
M.5.7	de.denkselbst.matrix.Matrix.SatzInfoDialog.java	459
M.5.8	de.denkselbst.matrix.Matrix.ShowTags.java	461
M.5.9	de.denkselbst.matrix.Matrix.HTMLKocher.java	463
M.5.10	de.denkselbst.matrix.Matrix.AbstandInfoDialog.java	465
M.5.11	de.denkselbst.matrix.Matrix.ShowAbstand.java	467
M.5.12	de.denkselbst.matrix.Matrix.WriteRandomMatrix.java	470
M.5.13	de.denkselbst.matrix.Matrix.Convert.java	470
M.5.14	de.denkselbst.matrixAligner.MatrixAligner.java	472
M.5.15	de.denkselbst.matrixAligner.AlignmentObservable.java	476
M.5.16	de.denkselbst.matrixAligner.AlignmentObserver.java	476
M.5.17	de.denkselbst.matrixAligner.AStarMatrixAligner.java	476
M.5.18	de.denkselbst.matrixAligner.BestCellMatrixAligner.java	486
M.5.19	de.denkselbst.matrixAligner.LMEDMatrixAligner.java	489
M.5.20	de.denkselbst.matrixAligner.BFSMatrixAligner.java	492
M.5.21	de.denkselbst.kommandozeile.Align.java	500
M.5.22	de.denkselbst.kommandozeile.HeadingMapper4Arno.java	502
M.5.23	de/denkselbst/kommandozeile/manifest	503
M.5.24	de/denkselbst/kommandozeile/manifest2	503
M.5.25	de.denkselbst.beobachtung.Beobachter.java	503
M.5.26	de.denkselbst.beobachtung.Beobachtbar.java	503
M.5.27	de.denkselbst.mavis.Mavis.java	503
M.5.28	de.denkselbst.mavis.MACV.java	512
M.5.29	de.denkselbst.mavis.StutzenDialog.java	515
M.5.30	de.denkselbst.mavis.ShowHTMLFrame.java	517
M.5.31	de/denkselbst/mavis/manifest	518
<b>N</b>	<b>Phrasenalignment – Code</b>	<b>519</b>
N.1	Python-Paket Phrase	519
N.1.1	__init__.py	519
N.1.2	getsegs.py	519
N.1.3	zuordnung2.py	520
N.1.4	tags_de.txt	523

N.1.5	tags_en.txt . . . . .	532
N.1.6	phrasealign.py . . . . .	537
N.1.7	demo-zuord2.py . . . . .	540
N.1.8	mogel.py . . . . .	541
N.1.9	web_phrases.py . . . . .	542
<b>O</b>	<b>Datenbank – Code</b>	<b>545</b>
O.1	SQL-Skripte zum Erstellen der Datenbank . . . . .	545
O.1.1	db.sql . . . . .	545
O.1.2	index.sql . . . . .	546
O.1.3	clean.sql . . . . .	547
O.1.4	mrproper.sql . . . . .	547
O.2	Python-Datenbank-Schnittstelle . . . . .	547
O.2.1	__init__.py . . . . .	548
O.2.2	RegionVonDB . . . . .	548
O.2.3	Wörterbücher . . . . .	554
O.2.4	Datenbankserver und Hilfsklassen . . . . .	568
O.3	Korpora und die Datenbank . . . . .	623
O.3.1	korpus2db.py . . . . .	623
O.3.2	delkorpus.py . . . . .	626
<b>P</b>	<b>Demo-Code</b>	<b>628</b>
P.1	Code für die Demo-Application . . . . .	628
P.1.1	Java-Applet KoksAppe.java . . . . .	628
P.1.2	hit1e.rb . . . . .	632
P.1.3	hit2e.rb . . . . .	635
	<b>Literaturverzeichnis</b>	<b>638</b>
	<b>Index</b>	<b>639</b>

# Abbildungsverzeichnis

3.1	KBrowser – Baum-Ansicht . . . . .	26
3.2	KBrowser in Aktion . . . . .	27
3.3	KBrowser – Listen-Ansicht . . . . .	28
5.1	Ein Text aus dem de-News-Korpus im Originalzustand . . . . .	37
5.2	Ein Text aus dem NATO-Korpus im Originalzustand . . . . .	38
5.3	Ein Beispiel für das Ausgabeformat . . . . .	39
7.1	Ausschnitt aus dem parallelen Korpus “DE-News” . . . . .	49
7.2	“DE-News”–Ausschnitt maschinell alignt . . . . .	50
7.3	anderer Ausschnitt aus dem parallelen Korpus “DE-News” . . . . .	51
7.4	maschinell erstelltes Alignment des zweiten “DE-News”–Ausschnittes . . . . .	52
7.5	Korrespondenzen . . . . .	53
7.6	Abstandsmatrix und Alignment-Pfad . . . . .	54
7.7	Positivbeispiele mit Abstand null zur Übersetzung . . . . .	57
7.8	Grafische Darstellung einer Abstandsmatrix . . . . .	69
7.9	Visualisierungstool “Mavis” . . . . .	72
7.10	Ausschnitt aus der Abstandsmatrix zum besprochenen Dokument . . . . .	74
8.1	Ausgangssegment . . . . .	79
8.2	Schritt 1: Wörter mit irrelevanten Tags streichen . . . . .	79
8.3	Schritt 2: Übersetzungen markieren . . . . .	79
8.4	Schritt 3: Mithilfe von Tagfolgen Phrasen erkennen . . . . .	80
8.5	Schritt 4: Tagfolgen einer Kategorie einander zuordnen . . . . .	80
9.1	Das Datenbankdesign und die Verzeigerung veranschaulicht . . . . .	86
10.1	Die leere Demoapplikation. . . . .	91
10.2	Das Wort ”hat” wurde markiert. . . . .	92
10.3	Es wurde links unten die gefundene Kollokation ”hat nichts zu sagen” ausgewählt. . . . .	93
10.4	Kommunikation . . . . .	93
12.1	Arbeitszeit je Woche ab dem 16. Oktober 2000 . . . . .	108
F.1	Zusammenspiel der einzelnen Alignment-Tools . . . . .	225
F.2	Mehrere Matrizen in einer Datei . . . . .	231
F.3	Korrekte Alignmentdatei . . . . .	232

# Tabellenverzeichnis

3.1	Korpora: Größen (Speicherplatz) . . . . .	19
3.2	Korpora: Wörter und Zeichen . . . . .	19
3.3	Korpora: Segmente in getaggtten Daten . . . . .	19
3.4	Korpora: Tokens in getaggtten Daten . . . . .	20
7.1	Church-und-Gale-Abstandsmaß . . . . .	56
7.2	Trigramme des Wortes 'menschenscheu' . . . . .	57
7.3	Positivbeispiele mit kleinem Abstand . . . . .	58
7.4	Positivbeispiele mit großem Abstand . . . . .	59
7.5	Nicht zuordbare Beispiele mit großem Abstand . . . . .	59
7.6	Negativbeispiele . . . . .	60
7.7	Trigramme als Hinweis auf Wortentsprechungen . . . . .	61
7.8	Weitere mehrfach auftretende Trigramme . . . . .	62
7.9	Häufigkeit einiger Wörter mit POS-Tag 'IN' in Satzpaaren, die 'mit' enthalten . . . . .	63
7.10	Relevante Wörter und gefundene Übersetzungen . . . . .	64
7.11	Abstandsmaße im KoKS-System . . . . .	65
7.12	Verteilung der Trigrammabstände . . . . .	66
7.13	Verteilung der Trigrammabstände zur sechsten Potenz . . . . .	66
7.14	Verteilung der kombinierten Abstände . . . . .	67
7.15	Kombinierten Abstände ohne Bonusausnahme . . . . .	67
7.16	Varianten des Programms <code>align.sh</code> . . . . .	71
7.17	Der Alignmentfehler aus Abbildung 7.4 . . . . .	74
8.1	Häufigkeit der Belege pro Phrase . . . . .	81
8.2	Beispiele für erkannte Phrasen . . . . .	82

# Vorwort

Der vorliegende Bericht dokumentiert das Ergebnis des Studienprojekts KoKS – „Korpusbasierte Kollokationssuche“. Von Oktober 2000 bis November 2001 haben wir uns gemeinsam auf die Suche nach Kollokationen im Deutschen und Englischen gemacht. Rückblickend erscheinen so manche Probleme, die uns im Projektverlauf Kopfzerbrechen bereiteten, einfach, während andere Aufgaben weiterhin auf eine Lösung warten.

Die vielen kleinen persönlichen Hochs und Tiefs, produktive und ergebnislose Projektsitzungen, technische Hürden und erfolgreiche Präsentationen werden uns sicher in Erinnerung bleiben. Wenn auch nicht alle Ziele erreicht worden sind, blicken wir doch mit etwas Stolz auf das Geleistete zurück.

An dieser Stelle sei auch ein herzliches Dankeschön an Petra Ludewig und Helmar Gust gesagt, die uns während des Projekts als Betreuer mit Rat und Tat zur Seite standen.

Osnabrück, November 2001

ARNO ERPENBECK

BRITTA KOCH

NORMAN KUMMER

PHILIP REUTER

PATRICK TSCHORN

JOACHIM WAGNER

**Teil I**

**Einleitung**



# Kapitel 1

## Überblick

### Inhaltsverzeichnis

---

1.1 KoKS: Ziele und Ideen . . . . .	4
1.2 Architektur . . . . .	5

---

## 1.1 KoKS: Ziele und Ideen

### 1.1.1 Die Motivation hinter KoKS

KoKS steht für „**K**orpusbasierte **K**ollokations-**S**uche“. Das Phänomen „Kollokation“ ist seit langem bekannt; vor allem Lexikographen und Fremdsprachdidaktiker beschäftigen sich damit. Ein Fremdsprachlerner beginnt beim Lernen neben der Grammatik meist mit Vokabel. Irgendwann hat er dann einen Stand erreicht, mit dem er sich in der Fremdsprache verständlich ausdrücken kann. Doch „verständlich“ heißt nicht „adäquat“: Wer eine Fremdsprache gewandt und sicher wie ein Muttersprachler einsetzen möchte, muss Kollokationen beherrschen: mehr oder weniger feste Mehrwortausdrücke, die meist nicht kompositionell übersetzbar sind <sup>1</sup>. *Starker Raucher* beispielsweise ist ein solcher Mehrwortausdruck:

(1.1) Peter ist ein starker Raucher.

(1.2) \*Peter is a strong smoker.

(1.3) Peter is a heavy smoker.

Um den propositionellen Inhalt von (1.1) auf Englisch wiederzugeben, genügt es nicht, die Übersetzungen der einzelnen Wörter, etwa *strong* als die von *stark*, zu kennen: (1.2) drückt nicht dieselbe Proposition aus. Wichtig ist zu wissen, dass ein *starker Raucher*, also jemand, der viel raucht, auf Englisch ein *heavy smoker* ist; (1.3) gibt den Inhalt von (1.1) richtig wieder.

Für Fremdsprachdidaktiker ist hier die Frage: „Wie lernt ein Fremdsprachlerner Kollokationen?“ Eine weit verbreitete, auch vom kognitiven Aspekt her gesehen interessante Ansicht ist: Kollokationen müssen wie Vokabeln gelernt werden. Problematisch ist hierbei, dass neben der eigentlichen Kollokation auch deren Flexibilität mit gelernt werden muss. Manche Kollokationen sind recht feste Wortkombinationen, die weder von der Wortstellung noch syntaktisch (z.B. in einem anderen Tempus) verändert werden können, ohne dass sie ihre besondere Funktion verlieren. Andere sind dagegen sogar erweiterbar.

Lexikographen beschäftigt vor allem die Frage: „Welche Kollokationen müssen im Lexikon verzeichnet werden?“ Das ist eine Frage, die zwischen notwendiger Information und Idiosynkrasie auf der einen und Platzbeschränkungen und Handhabbarkeit auf der anderen Seite navigiert.

---

<sup>1</sup>Zum Kollokationsverständnis von KoKS und zu benachbarten Begriffen wie z.B. „Idiom“ vgl. Kap. 2

### 1.1.2 Ziele von KoKS

In beiden Fällen soll der Ansatz von KoKS Lücken füllen. Das Ziel von KoKS ist der Aufbau einer Datenbank, die zweisprachige (englisch-deutsche) Phrasenpaare enthält, sowie die Entwicklung eines Prototypen, der – darauf aufbauend – Kollokationen erkennt. Ist nämlich eine solche Datenbank vorhanden, liegt eine umfangreiche „Vokabelliste“ vor, wobei die „Vokabeln“ hier Phrasen, insbesondere kollokative Phrasen, sind. Das Problem ist, wie eine solche Datenbank erstellt werden kann, ohne viele Lexikographen und anderen Linguisten unnötig lange zu beschäftigen, also: wie eine solche Datenbank mehr oder weniger automatisch gefüllt werden kann. Das KoKS-System nutzt hierfür parallele Korpora. Ein Korpus ist eine Menge von Texten. In einem parallelen Korpus liegen diese Texte in zwei Sprachen vor, für KoKS sind dies deutsche Texte mit ihrer englischen Übersetzung. Die Idee, die damit verbunden ist, lautet: In den parallelen Korpora werden zunächst Paare von Sätzen, die Übersetzungen voneinander sind, ermittelt; anschließend werden diese Paare in syntaktische Phrasen gesplittet, zu denen dann die Übersetzung im korrespondierenden Satz gesucht wird. Ein solches Phrasenpaar wird in eine Datenbank eingetragen. Kollokationen werden in KoKS als eine bestimmte Teilmenge dieser Phrasen aufgefasst: Phrasen nämlich, bei denen eine „kompositionelle“ Übersetzung fehlschlägt, d.h. die (meisten) Übersetzungen der in einer Phrase enthaltenen Wörter tauchen nicht in der korrespondierenden Phrase der anderen Sprache auf.

Hierfür sind datenintensive Methoden vonnöten, denn KoKS verfügt über kein tiefes linguistisches Wissen. Ein einziger Beleg für die Übersetzung einer Phrase ist weniger bedeutsam als ein Phrasenpaar, das häufig im Korpus auftaucht. Große parallele Korpora und statistische Verfahren sollen gewährleisten, dass Kollokationen korrekt erkannt werden.

### 1.1.3 Anwendungen

Mit diesem zentralen Ziel, dem Aufbau der Datenbank, sind weitere Ziele verbunden. Die Datenbank kann nun zu bestimmten ihr bekannten Phrasen passende Übersetzungen liefern. Um dies zu nutzen, ist im KoKS-Projekt ein Prototyp entwickelt worden. Dieser Prototyp soll in einem beliebigen Satz einer der beiden Sprachen Deutsch und Englisch kollokative Phrasen erkennen, passende Übersetzungen liefern und diese mit Verwendungsbeispielen veranschaulichen.

So kann ein Fremdsprachenlerner auf einfache Weise einerseits eine Übersetzungshilfe für nicht verstandene Phrasen erhalten, andererseits gibt ihm das System die Möglichkeit, zu kollokativen Ausdrücken seiner Muttersprache das adäquate Korrelat der Fremdsprache zu finden. KoKS kann ihm also bei Sprachrezeption und -produktion unterstützen. Und KoKS ist an dieser Stelle nicht auf Kollokationen beschränkt: Gerade weil dieser Begriff so schwer einzugrenzen ist, kann das System zu beliebigen in den parallelen Korpora vorhandenen Phrasen eine Übersetzung und Verwendungsbeispiele liefern, unabhängig von der „Kollokativität“.

## 1.2 Architektur

Das Ziel von KoKS ist der Aufbau einer Datenbank mit Wort- und Phrasenpaaren: eine deutsche Phrase mit ihrem englischen Korrelat. Auf dieser Basis können dann aus einem gegebenen Satz Wörter und Phrasen extrahiert werden, zu denen eine passende „Übersetzung“ nachgeschlagen werden kann; zu jeder Teilphrase sind dann auch Verwendungsbeispiele aus den parallelen Korpora vorhanden. Kollokative Phrasen werden anhand des Fehlens einer wörtlichen Übersetzung und mit Methoden der Statistik identifiziert.

Die Idee, wie dies geschehen kann, ist folgende: Phrasenpaare werden aus parallelen Korpora, also großen Textbeständen in deutsch und der englischen Übersetzung (oder umgekehrt), extrahiert. Dazu werden erst Sätze, dann Phrasen einander zugeordnet; der Fachbegriff aus der Computerlinguistik (CL) lautet „Alignment“. Das Alignment geschieht in KoKS auf der Basis von lexikalischem Wissen (nämlich mit Hilfe von Wortübersetzungen). Aus diesem Grund benötigt KoKS auch zweisprachige Wörterbücher (WB). Hier liegen bereits alignierte Wortpaare vor, die als Initialphrasen in die Datenbank eingetragen werden.

Folgende Verarbeitungsschritte sind notwendig:

- Normalisierung der Korpora
- Normalisierung der WB (Dictionary Entry Parsing, DEP)
- Tagging der WB-Einträge und Korpora

- Satzalignment der Korpora
- Phrasenerkennung und Aufbau von Phrasenkorrespondenzen
- Eintrag neuer Phrasenpaare in die Datenbank

**Normalisierung** Die in KoKS verwendeten Korpora liegen in verschiedenen Formaten vor. Die Spanne reicht von gutem bis schlechtem HTML über PDF, reinem ASCII bis zu proprietären Formaten. Damit der Dokumenttext verarbeitet werden kann, ist eine Normalisierung der Texte notwendig. Diese werden im Prinzip in ein annotiertes Textformat überführt.

**Dictionary Entry Parsing** Ähnliches gilt für die Wörterbücher: Dem KoKS-System liegen verschiedene Wörterbücher zu Grunde. Alle verwenden unterschiedliche Formate: etwa verschiedene Delimiter, um ein Lemma von seiner Übersetzung zu trennen, oder verschieden Arten, um grammatische Informationen zu kodieren.

Für KoKS ist im Wesentlichen eine Liste von Wortpaaren interessant: ein deutsches Lemma und dessen englische Übersetzung. Solche Listen müssen für jedes Wörterbuch(format) anders extrahiert werden; der Fachbegriff aus der CL lautet „Dictionary Entry Parsing“.

**Tagging** Sowohl die Wörterbücher als auch die Korpustexte werden nach der Normalisierung mit einem Part-of-Speech-Tagger getaggt. Die dadurch gewonnene Wortartinformation hilft bei der Phrasenerkennung und -zuordnung: Das Tagergebnis liefert die Grundlage für ein rudimentäres Chunk-Parsing.

**Satzalignment** Die Wörterbucheinträge sind „von Haus aus“ optimal align; die parallelen Korpora dagegen nicht. Der von uns entwickelte Satzaligner ordnet Sätze einander zu, meist einem deutschen einen englischen Satz, nicht selten aber korrespondiert ein deutscher Satz mit zwei englischen; auch Drei-zu-zwei-Alignment kommt vor. Das Alignment ist essenziell für KoKS: Ohne diese Information kann die Phrasenzuordnung nicht durchgeführt werden.

Der KoKS-Aligner nutzt, im Gegensatz zu vielen anderen Satzalignern, lexikalisches Wissen. Ein Aligner berechnet die Korrespondenz von Sätzen mit Hilfe eines Abstandsmaßes, das angibt, wie weit zwei Sätze „voneinander entfernt“ sind, wie groß also ihre Zugehörigkeit zueinander ist. Das Abstandsmaß in KoKS basiert auf Wortübersetzungen, die der Aligner in den Wörterbüchern findet.

**Phrasenerkennung und -alignment** Sind die Sätze einander zugeordnet, geht es in die entscheidende Phase: Phrasenerkennung und -alignment. Hier wird zunächst das durch das Tagging vorliegende Wortartwissen genutzt, um Phrasenhypothesen zu generieren. Anschließend wird zu den möglichen Phrasen ein Gegenstück im entsprechenden Satz der anderen Sprache gesucht. Das geschieht mit Rückgriff auf bereits eingetragene Phrasen- und Wortpaare. Auf diese Art gefundene Phrasenpaare werden in die Datenbank eingetragen; sei es als Neueintrag, sei es als weiterer Beleg für ein Phrasenpaar.

**Demo-Applikation** In diese Architektur gliedert sich auch eine webbasierte Applikation ein. Ein Fremdsprachenlerner hat hier die Möglichkeit, einen unverständenen Satz einzugeben (per copy & paste oder per Tastatur) und auf ein Wort zu klicken, das seiner Meinung nach etwas mit seinen Verständnisschwierigkeiten zu tun hat. KoKS liefert dann aus dem Satz extrahierte Phrasen und Verwendungsbeispiele.

Dazu wird der Input zunächst getaggt, um die zur Phrasenerkennung nötige Wortartinformation zu erlangen. Der Schritt des Satzalignments ist hier überflüssig, denn es wird ja nur eine Sprache verarbeitet. Wie beim Füllen der Phrasendatenbank werden in der getaggtten Eingabe Phrasenhypothesen generiert, um den Satz in Phrasen zu zerlegen. Für das angeklickte Wort und die möglichen Phrasen, in denen es enthalten ist, wird nun in der Datenbank nach einer Übersetzung gesucht. Bei Bedarf (d.h. wenn der Benutzer es anfordert) werden zu einer Phrase auch Verwendungsbeispiele aus den Korpora geliefert.

# Kapitel 2

## Kollokationen

### Inhaltsverzeichnis

---

2.1 Von Wortkombinationen zu Kollokationen . . . . .	7
2.2 Kollokationen in der Sprachwissenschaft . . . . .	9
2.3 KoKS und Kollokationen . . . . .	12

---

### 2.1 Von Wortkombinationen zu Kollokationen

Sprecher und Schreiber einer Sprache kombinieren die lexikalischen Einheiten – die Wörter – zu Äußerungen und Sätzen. Solche Kombinationen unterliegen verschiedenen, z.B. syntaktischen und semantischen Beschränkungen. Bestimmte Wortkombinationen innerhalb einer Äußerung scheinen in gewisser Weise besondere, ausgezeichnete Kombinationen zu sein: Hier bilden mehrere Wörter eine neue lexikalische Einheit, sogenannte *Mehrwortlexeme*; die Wörter gehören „irgendwie“ fester zusammen. Syntaktische und semantische Selektionskriterien allein reichen zur Begründung dieses Zusammengehörens nicht aus. Dabei hat die Festigkeit solcher Wortkombinationen einen graduellen Charakter. Eine Reihe von Beispielen soll dies verdeutlichen.

(2.1) In der Innenstadt wird ein Haus gebaut.

(2.2) Maria hat gerne einen Strauß schöner Blumen im Wohnzimmer stehen.

Die Wörter in (2.1) sind frei kombiniert, d.h. sie sind auf keine besondere Weise aneinander gebunden; ihre Auswahl aus der Menge aller im Deutschen zur Verfügung stehenden Wörter erfolgt aus rein semantischen und syntaktischen Gründen: Mit ihnen soll eine bestimmte Proposition zum (syntaktisch wohlgeformten) Ausdruck gebracht werden. Neben diesen Kriterien spricht nichts für die Auswahl genau dieser lexikalischen Einheiten. Die Nomen-Verb-Kombination (*ein*) *Haus gebaut* lässt sich genauso gut durch (*ein*) *Gebäude errichtet* ersetzen: *Haus* und *bauen* sind durch die Synonyme *Gebäude* bzw. *errichten* ausgetauscht<sup>1</sup>. Durch den Austausch der Wörter behält der Satz seine Bedeutung, und er klingt für einen kompetenten Sprecher des Deutschen auch keineswegs ungewöhnlich.

Analog verhält es sich in (2.2). Auch hier lässt sich die Wortkombination durch semantische und syntaktische Kriterien erklären. In der Adjektiv-Nomen-Kombination *schöne Blumen* kann *schön* durch *hübsch* ersetzt werden, ohne dass sich an Sinn und Gebräuchlichkeit etwas ändert.

Betrachten wir aber folgende Beispiele:

(2.3) Du solltest mal deine Schuhe zumachen.

(2.4) Hast du dir schon die Zähne geputzt?

---

<sup>1</sup>Die Synonymie der genannten Begriffe ist sicher diskussionsfähig; an dieser Stelle soll es aber um „Gleichbedeutung“ im weiteren Sinne gehen

Auf den ersten Blick unterscheiden sich diese Beispiele hinsichtlich ihrer Wortkombinationen nicht von (2.1) und (2.2). Versucht man aber, einzelne Wörter durch Synonyme zu ersetzen, so trifft man auf ein interessantes Phänomen: Ersetzt man in (2.3) *zumachen* durch das Synonym *schließen* (vgl. etwa *Kannst du mal die Tür zumachen/schließen?*), so erhält man *Du solltest mal deine Schuhe schließen*. Der Satz ist sicher verständlich, einem kompetenten Sprecher des Deutschen wird er aber ungewöhnlich erscheinen: Man sagt *Schuhe zumachen* und nicht *Schuhe schließen*.

Das Gleiche tritt auch in (2.4) auf: Die Ersetzung von *putzen* durch *sauber machen* führt zum verständlichen, aber ungebräuchlichen *Hast du dir schon die Zähne sauber gemacht?* Interessant ist auch die Frage nach der Übersetzung in eine andere Sprache, etwa das Englische: *Putzen* wird meist mit *clean* übersetzt. *Sich die Zähne putzen* gibt der Briten aber mit *to brush one's teeth* wieder. Die spezielle Verwendung von *brush* für *putzen* wird auch dadurch deutlich, dass es zwar im Deutschen analog *sich die Nase putzen* heißt, im Englischen aber nicht *\*to brush one's nose*, sondern *to blow one's nose*; im Deutschen heißt es umgekehrt auch nicht *sich die Zähne bürsten*.

(2.5) Peter ist ein starker Raucher.

(2.6) Hans ist ein guter Esser.

In (2.5) ist die Kombination *starker Raucher* von Interesse. Ersetzt man hier *stark* durch *kräftig*, so erhält man den Satz *Peter ist ein kräftiger Raucher*. Diese Formulierung ist nicht nur ungebräuchlich, sondern gibt vor allem gar nicht die Bedeutung von (2.5) wieder. *Stark* hat hier also eine vom normalen sprachlichen Kontext abweichende Bedeutung, (2.5) bedeutet in etwa, dass Peter verhältnismäßig viele Zigaretten raucht. Diese Bedeutung von *stark* ist weniger an diesen speziellen semantischen Kontext als vielmehr an das Wort *Raucher* gebunden: (2.6) drückt analog aus, dass Hans verhältnismäßig viel Nahrung zu sich nimmt; er ist aber kein *starker Esser*, sondern eben ein *guter Esser*, und auch *gut* hat eine besondere Bedeutung. In diesen Beispielen finden wir also Adjektiv-Nomen-Kombinationen, die sich nicht rein semantisch begründen lassen. Ihre Partner gehören auf eine bestimmte Weise eng zusammen.

Ähnlich sind folgende Beispiele geartet:

(2.7) Der Kanzler hielt auf dem Gewerkschaftertreffen eine bemerkenswerte Rede.

(2.8) Wir warten lieber, bis sein Zorn verraucht ist.

In diesen Beispielen beinhalten die Kombinationen *eine Rede halten* und *Zorn verraucht* jeweils ein Nomen, das seine ursprüngliche Bedeutung trägt (*Rede* bzw. *Zorn*) und ein Verb, das eine von normalen Kontexten abweichende Bedeutung trägt (*halten* bzw. *verrauchen*). Während es sich in (2.5) und (2.6) um Adjektiv-Nomen-Kombinationen handelt, sind in (2.7) eine Verb-Nomen-Kombination („Rede“ ist hier Objekt) und in (2.8) eine Nomen-Verb-Kombination („Zorn“ ist hier Subjekt) solche fester zusammengehörenden Kombinationen.

Als abschließende Beispiele seien die folgenden genannt:

(2.9) Leider ist die Party ins Wasser gefallen.

(2.10) Hans hat an Maria einen Narren gefressen.

In (2.9) ist die Kombination *ins Wasser fallen* eine sehr feste Kombination. Keines der Wörter hat seine ursprüngliche Bedeutung, und keines der Wörter lässt sich durch ein synonymes austauschen: *in das Wasser gefallen* (*ins* durch *in das* ersetzt), *ins Meer gefallen* (*Wasser* durch *Meer* ersetzt) und *ins Wasser geplumpst* (*fallen* durch *plumpsen* ersetzt) haben nicht mehr die übertragene, sondern eine wörtliche Bedeutung, also eine, die sich kompositionell aus der Bedeutung der Bestandteile zusammensetzt. Die Kombination kann auch nicht erweitert werden: *ins kalte Wasser gefallen* trägt ebenso nicht mehr die übertragene Bedeutung. Eine Variation des Tempus ist jedoch möglich: *Die Party wird wohl ins Wasser fallen*.

Ähnlich fest ist auch (2.10). *An jemandem einen Narren gefressen haben* ist eine sehr unflexible Kombination: Selbst eine Variation des Tempus ergibt einen unverständlichen Satz: *Hans frisst an Maria einen Narren*.

Wie fest eine Wortkombination ist, kann also stark variieren. Die Pole dieses Kontinuums bilden auf der einen Seite sogenannte *freie Kombinationen*, wie sie in (2.1) und (2.2) auftreten. Auf der anderen Seite stehen die sehr festen Kombinationen wie in (2.9) und (2.10), die sogenannten *Redensarten* oder *Idiome*. *Kollokationen* befinden sich dazwischen. Dabei sind die Grenzen schwer zu ziehen, und je nach Interessenlage und Zielsetzung ist der Kollokationsbegriff weiter oder enger gefasst.

## 2.2 Kollokationen in der Sprachwissenschaft

Im Zusammenhang mit diesem Phänomen des Zusammengehörens treten einige Fragen auf: Wie lässt sich der Begriff *Kollokation* klar beschreiben? Was sind Kollokationen, was nicht<sup>2</sup>? Wie kann das Zustandekommen einer solch festen Kombination erklärt werden? Wie können Kollokationen (automatisch) bestimmt werden? Wie sind Kollokationen vom lexikographischen Standpunkt aus zu behandeln? Welche Rolle spielen Kollokationen beim (Fremd-)Spracherwerb?

Im Folgenden soll ein Einblick in linguistische Ansätze zum Umgang mit Kollokationen gegeben werden. Es kann sich hierbei nicht um einen vollständigen Überblick handeln, vielmehr soll die große Bandbreite bei der Definition von Kollokationen angedeutet werden, die zum großen Teil aus unterschiedlichen Zielsetzungen bei der Beschäftigung mit Kollokationen resultiert. Insbesondere sollen auch nicht alle o.g. Fragen beantwortet werden.

### 2.2.1 Kollokationen bei Firth

Der Begriff *Kollokation* wurde von *John R. Firth* 1957 in die sprachwissenschaftliche Diskussion eingeführt. Firth ist zugleich der Begründer des sogenannten (*Britischen*) *Kontextualismus*, einer Forschungsrichtung innerhalb der Linguistik, die Sprache in ihren sprachlichen und außersprachlichen Kontexten analysiert. Dabei sind der Gegenstand der Untersuchungen im Gegensatz zur Transformationsgrammatik nicht ein theoretisches Sprachmodell, sondern tatsächlich produzierte Äußerungen. Wesentlicher Bestandteil der sprachwissenschaftlichen Forschung ist demnach korpusbasierte Sprachverarbeitung (vgl. Lehr, 1996).

Firth unterscheidet mehrere Ebenen der linguistischen Analyse; zwei davon sind *Grammar* und *Lexis*. Erstere enthält das Phänomen *colligation*. Bei *colligation* handelt es sich um syntagmatische Beziehungen zwischen grammatischen Kategorien. Auf der Ebene der *Lexis* ist die *collocation* angesiedelt. „Collocation, for Firth, refers to the syntagmatic relations into which lexical items habitually enter, as is concerned as a part of the meaning of the lexical items concerned“ (Butler, 1985, S. 7). Als Beispiele nennt Firth (1957) *silly + ass, milk + cow, dark + night*. Eine der Bedeutungen von *night* ist demnach seine Kollokabilität mit *dark* und umgekehrt.

Was Firth genau mit Kollokabilität meinte, ist jedoch nie ganz klar geworden. Er hat sich vor allem programmatisch geäußert, und „viele seiner theoretischen Ansätze sind erst von seinen Schülern aufgenommen und aufgearbeitet worden“ (Bahns, 1996, S. 7). Offensichtlich versteht Firth unter *Kollokation* ein häufiges gemeinsames Vorkommen von zwei Begriffen, das sich hauptsächlich aus semantischen Gründen ergibt. Firths Verständnis von *Kollokation* hängt eng zusammen mit dem Begriff *Kookkurrenz*. Gerade durch den Gedanken des häufigen gemeinsamen Auftretens besteht eine Verbindung zu einem *statistisch orientierten Kollokationsverständnis*.

### 2.2.2 Automatische Gewinnung von Kollokationen mit statistischen Verfahren

Im Sinne von Firth gibt es Wörter, die besonders häufig zusammen in einem Text auftreten. Diesen Ansatz verfolgt *John Sinclair*, ein Schüler Firths. Auch Sinclair beschreibt mehrere Sprachebenen, u.a. auch *Grammatik* und *Lexikon*. Ziel seiner Arbeit ist es, die wichtigsten *patterns* einer sprachlichen Ebene zu beschreiben. Bei der Sprachproduktion gibt es auf der Ebene der Grammatik Wahlentscheidungen, die durch Kontraste bedingt sind, z.B. Aktiv/Passiv, Singular/Plural etc. Solche Kontraste gibt es auf der Ebene des Lexikons nicht. Auf dieser Ebene gilt das Interesse Sinclairs der Beschreibung nicht der semantischen, sondern der formalen Unterschiede lexikalischer Items. Die Ähnlichkeit zweier Lexeme wird nicht aufgrund semantischer Kriterien erstellt, sondern rein formal über Wörter, die in der Nähe der betrachteten Lexeme auftauchen, die also mit ihnen *kollokieren*. Grundlage solcher formaler Beschreibungen ist die Untersuchung möglichst großer Korpora.

Problematisch ist der Begriff der „Nähe“: Wie weit dürfen zwei Lexeme entfernt sein, um noch als Kollokationen zu zählen? Handelt es sich hier um ein binäres Maß (also „nah“ oder „nicht nah“) oder um ein Kontinuum (je näher sich zwei Lexeme sind, umso stärker kollokieren sie)? Sinclair entscheidet sich für ersteres und betrachtet zu einem Lexem, dessen Kollokationspartner gesucht sind – einem *node* –, alle Lexeme, die in einem (willkürlich festgelegten) Abstand davon – der *collocational span* – vorkommen. Mit dem *node* kollokierende Lexeme sind sogenannte *collocates*. Sinclair arbeitet in seinen Beispielanalysen

<sup>2</sup>An dieser Stelle sei angemerkt, dass „Kollokation“ sowohl das (abstrakte) Sprachphänomen als auch eine (konkrete) Wortkombination meinen kann.

mit einer *collocational span* von  $\pm 3$  und  $\pm 4$  Wörtern. Zu einem *node* werden dann die *collocates* gezählt. Sinclair unterscheidet *casual* und *significant* collocations. Für letztere ist nicht nur wichtig, wie häufig ein *collocate* mit dem *node* kolloziert; diese Häufigkeit wird auch in Bezug gesetzt zur Häufigkeit des *collocates* in der gesamten Untersuchungsbasis. Ein Wort, das besonders häufig innerhalb und nur selten außerhalb der *collocational span* eines anderen auftaucht, bildet gemeinsam mit diesem *node* eine *significant collocation*.

In Sinclair (1978) werden als Beispiel Kollokationen mit „back“ genannt. Bahns (1996) listet einige auf, u.a. (die Kollokationspartner von „back“ sind jeweils unterstrichen):

(2.11) Don't try to hold her *back*

(2.12) He leaned *back* in his chair

(2.13) He *leaned back* in his chair

(2.14) We had to go *back* to the hotel

(2.15) You must come *back* to the kitchen

Diese Ergebnisse, die auf automatisierten statistischen Verfahren beruhen, zeigen das recht weite Kollokationsverständnis Sinclairs. Viele Linguisten würden etwa (2.14) und (2.15) nicht als Kollokationen ansehen; „hotel“ und „kitchen“ stehen hier in keiner engen Beziehung zu „back“. Um solche Ergebnisse zu vermeiden, filtert etwa Kjellmer (1987) die statistisch gewonnenen Ergebnisse durch linguistische Filter. Er definiert:

„A collocation is a sequence of words that occurs more than once in identical form [...] and which is grammatically well structured.“ (Kjellmer, 1987, S. 133)

Demnach ist eine Kollokation also eine zusammenhängende Wortkombination, die mehr als einmal identisch im zu Grunde liegenden Korpus auftaucht und deren Bestandteile in einer syntaktischen Beziehung zueinander stehen. Letzteres untersucht Kjellmer manuell. Sein eher technisches Kollokationsverständnis führt dazu, dass etwa „had been“ oder „United States“ als Kollokationen klassifiziert werden. Diskontinuierliche Kombinationen und durch Tempuswechsel oder Passivierung umgestellte Kombinationen können dagegen nicht als Kollokation identifiziert werden. Das birgt insbesondere für das Deutsche, das eine sehr freie Wortstellung hat, Probleme.

Zweifellos bietet eine statistische Herangehensweise an Kollokationen viele Chancen: Statt sie mühsam von Hand aufzulisten, können Kollokationen automatisch extrahiert werden, und anstatt Beispiele selbst zu konstruieren, kann ein Linguist „real existierende“ Verwendungen aus großen Korpora gewinnen. Ein solches automatisiertes Verfahren geht aber eben einher mit einigen Nachteilen. So kritisiert Breidt (1995):

„Collocations in the sense of ‘frequently occurring words’ can quite easily be extracted from corpora by statistic means. From a linguistic point of view, however, a more restricted use of the term is preferable.“ (Breidt, 1995, S. 2)

Die Häufigkeit einer Wortkombination ist nicht unbedingt ein gutes Kriterium für Kollokationen. Auch die Forderung nach identischer Wiederholung im Korpus und nach einer zusammenhängenden Kombination beschneidet den Kollokationsbegriff an der falschen Stelle: Viele Kollokationen können z.B. im Tempus variiert werden; oft kann eine Kollokation auch erweitert werden (z.B. eine Verb-Nomen-Kollokation durch ein das Nomen modifizierendes Adjektiv)<sup>3</sup>. Ob und inwieweit ein statistischer Ansatz sinnvoll ist, hängt also sehr stark von der jeweiligen Zielsetzung ab.

### 2.2.3 Kollokationen in Lexikographie und Fremdsprachdidaktik

Schon lange spielen Kollokationen eine große Rolle in der Lexikographie. Hier besteht die Frage, was ins Lexikon aufgenommen werden sollte und was nicht. Lexikographen befinden sich in einem Konfliktfeld: Ein Lexikoneintrag sollte so ausführlich wie nötig sein und gleichzeitig so knapp wie möglich. Ein zu umfangreiches Lexikon ist (zumindest in einer gedruckten Form) zu unübersichtlich und schwer zu handhaben; insbesondere Redundanzen müssen daher vermieden werden. Auf der anderen Seite soll ein Nutzer des Lexikons alle Informationen finden, die er braucht.

<sup>3</sup>Kober (2000) geht ausführlich auf Variationsmöglichkeiten u.a. von Kollokationen ein

Anthony P. Cowie befasst sich beispielsweise mit der Kollokationsproblematik, um praktische Probleme zu lösen, die bei der Arbeit am *Oxford Dictionary of Current Idiomatic English (ODCIE)* auftraten (vgl. Cowie, 1983). Hierbei geht es laut Bahns (1996) darum, welche Wortverbindungen auf welche Weise in das ODCIE aufgenommen werden sollen. Der Begriff *Idiom* umfasst bei Cowie auch Kollokationen. Er unterteilt solche Wortverbindungen in vier Kategorien: *Pure idioms* tragen ausschließlich eine nicht-kompositionelle Bedeutung und sind nicht variierbar. *Figurative idioms* haben neben der übertragenen auch eine wörtliche Bedeutung und sind selten variierbar. *Restricted collocations* umfassen Kombinationen, in denen ein Bestandteil eine übertragene und einer seine ursprüngliche Bedeutung trägt; ein gewisses Maß an lexikalischer Variation ist möglich. *Open collocations* umfassen Kombinationen, deren Bestandteile ihre wörtliche Bedeutung tragen und die demnach frei kombinierbar sind. Bis auf die der letzten Kategorie sind alle diese Wortkombinationen ins Lexikon aufzunehmen.

Ist einmal entschieden, welche Wortverbindungen ins Lexikon aufgenommen werden sollen, bleibt eine weitere Frage bestehen: An welcher Stelle wird etwa eine Kollokation aufgeführt? Ein Lexikon ist in aller Regel alphabetisch sortiert. Dies führt aber bei Mehrworteinträgen zu Problemen: Unter welchem Lemma ist eine Kollokation zu finden? Aktuelle Lexika gehen hier unterschiedliche Wege, so dass ein Benutzer zum Auffinden von Kollokationen oft unter den Lemmata mehrerer Kollokationspartner nachschlagen muss.

In dieser Hinsicht ist die Arbeit von Franz Josef Hausmann interessant. Er vertritt einen syntaktisch orientierten Ansatz zur Beschreibung von Kollokationen und stellt fest, dass die Kollokationspartner funktional nicht gleichgestellt sind (s.u.).

Hausmann darf „innerhalb der Sprachwissenschaft in Deutschland [...] zweifellos als derjenige betrachtet werden, der sich am intensivsten mit dem Kollokationsbegriff auseinandergesetzt hat“ (Bahns, 1996, S. 22). Neben lexikographischen Aspekten der Kollokationsforschung befasst er sich auch mit der fremdsprachdidaktischen Problematik, die mit Kollokationen verbunden ist. Er erstellt (Hausmann, 1984) eine Typologie von Wortverbindungen. Er unterscheidet *fixierte* von *nicht-fixierten* Wortverbindungen; erstere umfassen Redewendungen und Komposita und sind als eine lexikalische Einheit aufzufassen. Letztere gliedern sich – je nach Grad der Kombinierbarkeit – in *Ko-Kreationen*, *Kollokationen* und *Konter-Kreationen*. *Ko-Kreationen* sind bezüglich ihrer Kombinierbarkeit nur durch semantische Restriktionen eingeschränkt. *Kollokationen* sind Verbindungen von Wörtern mit begrenzter Kombinierbarkeit; sie werden vom Sprecher im Gegensatz zu *Ko-Kreationen* nicht kreativ zusammengesetzt, sondern als „Halbfertigprodukt der Sprache“ (Hausmann, 1984, S. 398) als Kombination aus dem Gedächtnis abgerufen. *Konter-Kreationen* sind bewusste Verstöße gegen die Kombinierbarkeit zweier Wörter, etwa Celans „schwarze Milch“.

„Unter dem Gesichtspunkt der Üblichkeit sind die Kollokationen die Kombinationen von auffälliger Üblichkeit, die *Ko-Kreationen* solche von unauffälliger Üblichkeit und die *Konter-Kreationen* solche von auffälliger Unüblichkeit.“ (Hausmann, 1984, S. 399)

Hausmann hat also ein recht enges Kollokationsverständnis. Bei der Untersuchung von Kollokationspartnern unterscheidet er zwischen *Basis* und *Kollokator*. Eine Kollokation ist demnach eine Kombination aus einem semantisch autonomen Bestandteil, der innerhalb der Kollokation eine Bedeutung trägt, die er auch in *Ko-Kreationen* innehat – diesen Kollokationspartner nennt Hausmann *Basis* – und einem Bestandteil, dessen Bedeutung zumindest zum Teil nur mit Hilfe der *Basis* definiert werden kann – dem *Kollokator*. Hausmann unterteilt Kollokationen nach ihrer syntaktischen Struktur und nennt vier Klassen von Kollokationen:

1. Adjektiv-Nomen-Kollokationen, z.B. *eingefleischer Junggeselle*
2. Verb-Nomen-Kollokationen, z.B. *Zorn verbraucht* oder *eine Rede halten*
3. Adverb-Adjektiv-Kollokationen, z.B. *peinlich genau*
4. Adverb-Verb-Kollokationen, z.B. *verbissen kämpfen*

In allen diesen Klassen lässt sich eine *Basis* und ein *Kollokator* bestimmen. In 1 ist das Nomen die *Basis*, das Adjektiv der *Kollokator*: *eingefleischt* lässt sich nur mit Rückgriff auf *Junggeselle* vollständig definieren, während die Definition der Bedeutung von *Junggeselle* ohne einen solchen Rückgriff auskommt. In 2 ist ebenfalls das Nomen die *Basis*, das Verb ist der *Kollokator* (um die Bedeutung von *halten* vollständig

zu klären, muss die Kollokation *eine Rede halten* erwähnt werden). In Kollokationen des Typs 3 und 4 ist jeweils das Adverb der Kollokator, Adjektiv bzw. Verb sind die Basis.

Hieraus können sich Konsequenzen sowohl für die Lexikographie als auch die Fremdsprachdidaktik ergeben. Bei der Produktion einer Kollokation in einer Fremdsprache ist bei Lernern, die über ein ausreichendes Grundvokabular verfügen, die Übersetzung der jeweiligen Basis nicht das Problem; fraglich ist aber die des Kollokators. Die Übersetzung der Basis *Unfall* ins Englische wird dem Lerner bekannt sein, problematisch ist aber die der Kollokatoren, die mit *Unfall* eine Kollokation eingehen, etwa in *schwerer Unfall*, *ein Unfall ereignet sich* o.ä. Wortschatzarbeit muss daher Hilfe zur Produktion von Kollokationen geben, indem etwa zu einer Basis das Wortfeld der zugehörigen Kollokatoren erarbeitet wird. In der Lexikographie kann diese Unterscheidung zu einer klareren Systematik bezüglich der Verzeichnung von Kollokationen führen.

## 2.3 KoKS und Kollokationen

Das Projekt KoKS will Kollokationen durch Phrasenalignment paralleler Korpora automatisch extrahieren. Ein wichtiges Kriterium ist dabei die Übersetzbarkeit: Findet sich die kompositionelle Übersetzung einer deutschen Phrase nicht im englischen Gegenstück, so spricht dies dafür, dass es sich um eine Kollokation handelt. Als Indiz hierfür spricht: Ein Bestandteil der Kombination hat im Gegenstück der anderen Sprache ein Korrelat, das nicht im normalen Wörterbuch als Übersetzung vermerkt ist. Die Bedeutung dieses Wortes weicht also von der in normalen Kontexten ab. Ein solches Verständnis von Kollokation formuliert Breidt:

„[...] collocations shall refer only to word combinations with a lexically (rather than syntactically or semantically) restricted combinatory potential, where at least one component has a special meaning that it cannot have in a free syntagmatic construction.“ (Breidt, 1995, S. 2)

Dabei berührt die Arbeit in KoKS verschiedene der o.g. Ansätze. Das Interesse des Projekts gilt einerseits der automatischen Extraktion von Kollokationen. Die Idee, diese aus parallelen Korpora zu ermitteln, verspricht dabei, einige Probleme bisheriger automatischer Verfahren zu überwinden: Zwar fehlt auch KoKS ein tiefes semantisches Verständnis, um beispielsweise entscheiden zu können, ob ein Wort innerhalb einer bestimmten Kombination eine besondere Bedeutung trägt. Dies wird aber (wenigstens zu einem großen Teil) dadurch aufgefangen, dass zu jeder Verwendung eines Wortes im zu Grunde liegenden Korpus eine Übersetzung ins Englische vorliegt, anhand derer eine etwaige besondere Verwendung ermittelt werden kann.

Daneben verfolgt KoKS ein in der Fremdsprachdidaktik verankertes Ziel: Das System soll Hilfestellungen bieten zum Verständnis und zur Produktion fremdsprachlicher Wortkombinationen. Das ermöglicht übrigens, dass das Kollokationsverständnis in KoKS nicht auf eine scharfe Grenzziehung angewiesen ist: Die Frage „Handelt es sich bei einer bestimmten Wortkombination um eine Kollokation?“ muss nicht unbedingt entschieden werden; wichtig ist, durch die Angabe von Übersetzung und Verwendungsbeispielen dem Nutzer eine adäquate Ausdrucksmöglichkeit in der Fremdsprache zu ermöglichen.

Eine weitere Verbindung besteht natürlich auch zur Lexikographie. Da es sich bei KoKS aber um die Erstellung eines elektronischen Lexikons handelt, ist das Problem des beschränkten Umfangs einerseits und die Festlegung auf eine einzige Anordnung der Einträge andererseits nicht gegeben; für den Nutzer ist eine Schnittstelle zur Abfrage von Einträgen nötig, die möglichst flexibel, dabei aber auch einfach zu bedienen ist. Wie das Lexikon intern aufgebaut ist, spielt aus Sicht des Anwenders aber keine Rolle.

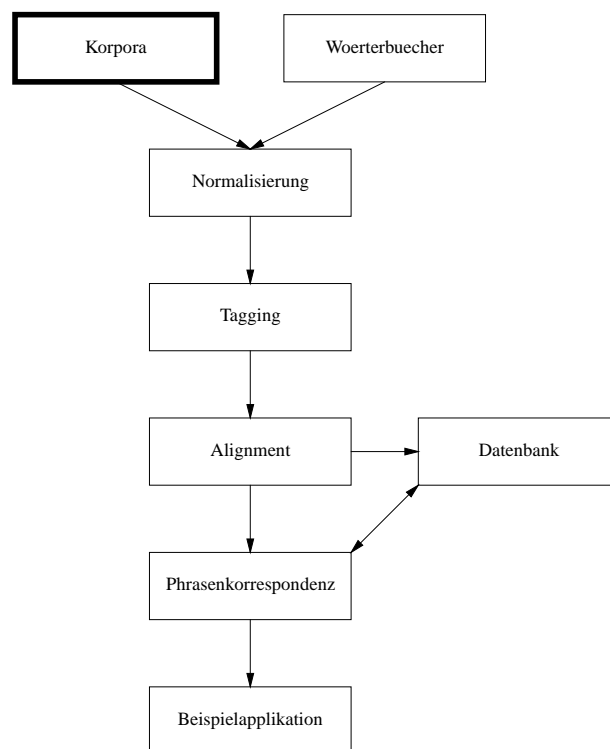
## **Teil II**

# **Linguistische Verarbeitung und Datenhaltung**



# Kapitel 3

## Korpora



### Inhaltsverzeichnis

---

3.1	Parallele Korpora . . . . .	16
3.2	Vorhandene Korpora . . . . .	16
3.3	Quantitative Angaben . . . . .	18
3.4	Aufbereitung . . . . .	20
3.5	Sprachidentifikation . . . . .	23
3.6	Verwaltung und Organisation . . . . .	24
3.7	Ausblick und Verbesserungen . . . . .	27

---

## 3.1 Parallele Korpora

Wie bereits in Abschnitt 1.1.2 beschrieben, stützt sich unser Ansatz auf parallele Korpora. Dieses Kapitel liefert einen Überblick über das KoKS-Korpus, dessen Aufbau und Verwaltung sowie einen Ausblick auf mögliche Verbesserungen. Tatsächlich sind parallele Korpora in Deutsch und Englisch nicht so frei verfügbar, wie man es gerne hätte. Will oder kann man die für kommerzielle Korpora üblichen Preise nicht zahlen, muss man sich überlegen, wie man aus dem frei erhältlichen Material brauchbare Daten zusammenstellen kann.

Zunächst liefert Abschnitt 3.2 eine Zusammenfassung über die von uns verwendeten Korpora. Anschließend werden in Abschnitt 3.3 einige quantitative Angaben über die Korpora gemacht. In Abschnitt 3.4 werden ein paar typische Probleme bei der Beschaffung und Aufbereitung der Daten nebst Lösungsvorschlägen geschildert. Eine besondere Aufgabe bei der Aufbereitung, nämlich die automatische Identifikation der Sprache eines Textes, wird in Abschnitt 3.5 behandelt. Schließlich präsentieren wir in Abschnitt 3.6 die Ideen und Programme zur Verwaltung der Korpus-Dateien; in Abschnitt 3.7 werden mögliche Verbesserungen diskutiert.

## 3.2 Vorhandene Korpora

### 3.2.1 Überblick

Dieser Abschnitt gibt einen Überblick über die von uns verwendeten Korpora. Da parallele Korpora leider nicht so frei verfügbar sind, wie man es gerne hätte, haben wir im Rahmen des KoKS-Projekts selbst einen Bestand an Texten in Deutsch und Englisch aufgebaut.

Ein guter Ausgangspunkt für die Suche nach Korpora ist die Webseite von Michael Barlow: <http://www.ruf.rice.edu/~barlow/corpus.html>. Des Weiteren findet man mit den gängigen Suchmaschinen Linksammlungen und Materialverweise.

Neben den Text-Dateien haben wir auch Wörterbücher mit in den Korpus integriert. Dadurch konnten alle „Text-Ressourcen“ einheitlich verwaltet werden. Eine detaillierte Beschreibung der Wörterbücher erfolgt in Kapitel 4.1.

#### Bibel

Als englische Version kommt die „King James Bible“ zum Einsatz, die deutsche Version folgt der Übersetzung von Martin Luther. Beide Sprachen beinhalten das Alte und das Neue Testament.

Bibel: Merkmale	
<b>Format</b>	HTML
<b>Quelle</b>	<a href="http://www.hti.umich.edu/index-all.html">http://www.hti.umich.edu/index-all.html</a>
<b>Art</b>	lange Texte
<b>Kategorie</b>	Religion

Das Alignment hat sich als sehr problematisch herausgestellt: Zum einen sind die beiden Fassungen nicht wirkliche Übersetzungen voneinander, sondern aus unterschiedlichen Quellen zu unterschiedlichen Zeiten entstanden. Zum anderen ist die Formatierung bzw. Struktur doch recht verschieden, was den Satzaligner manchmal aus dem Tritt bringt. Erschwerend kommt das recht spezielle Vokabular hinzu. In der späteren Projektphase haben wir die Texte der Bibel daher nicht mehr benutzt.

#### DE-News

Die DE-News bestehen aus mehreren Jahrgängen von Nachrichten-Texten. Die Original-Meldungen sind Deutsch und wurden von einem freiwilligen Projekt (also nicht von professionellen Übersetzern) ins Englische übersetzt. Eine Datei enthält in der Regel mehrere Meldungen eines Tages. Die Zuordnung der Nachrichten untereinander erfolgt durch vorhandene IDs.

DE-News: Merkmale	
<b>Format</b>	ASCII, Überschriften durch Tags gekennzeichnet
<b>Quelle</b>	<a href="http://www.isi.edu/~koehn/publications/de-news/">http://www.isi.edu/~koehn/publications/de-news/</a>
<b>Art</b>	kurze Texte
<b>Kategorie</b>	Politik, Sport

Bei genauerer Betrachtung der Texte haben sich eine Reihe kleinerer bis größerer Probleme herausgestellt, deren Behandlung in Abschnitt 3.4.2 vorgestellt wird. So war z.B. die Zuordnung der Nachrichten untereinander nicht immer fehlerfrei.

### EU

Der quantitativ größte Teil wurde von Texten aus dem Fundus der EU gestellt. Darin befinden sich u.a. Pressemitteilungen, Nachrichten-Texte und andere politische Dokumente wie Verträge. Die meisten Dokumente liegen in zahlreichen europäischen Sprachen vor, von denen für uns aber nur die deutsche und die englische Variante interessant sind.

EU: Merkmale	
<b>Format</b>	kaputtes HTML
<b>Quelle</b>	<a href="http://europa.eu.int/rapid/start/welcome.htm">http://europa.eu.int/rapid/start/welcome.htm</a>
<b>Art</b>	wechselnd, überwiegend kurz
<b>Kategorie</b>	Politik

Auch bei der Aufbereitung der EU-Texte kamen einige Widrigkeiten ans Tageslicht. Insbesondere stellte sich heraus, dass eine automatische Sprachidentifikation einiges erleichtern würde, siehe dazu Abschnitt 3.4.2. Ein bisher von uns nur wenig behandeltes Problem sind die zahlreichen Listen und Tabellen in den Dokumenten, siehe dazu Abschnitt 3.7.

### Linux

Eine reichhaltige Quelle schienen uns diverse Linux HOWTOs und FAQs. Allerdings wurden sie bisher nicht verwendet, weil die deutsche und englische Variante stark unterschiedliche Strukturen (z.B. zwei Kapitel Englisch entsprechen drei Kapiteln Deutsch) aufweisen.

Linux: Merkmale	
<b>Format</b>	HTML, ASCII
<b>Quelle</b>	<a href="http://www.linuxdoc.org/docs.html#howto">http://www.linuxdoc.org/docs.html#howto</a>
<b>Art</b>	längere Texte
<b>Kategorie</b>	Technik

### MCI

Aus dem MCI-Korpus wurden die Texte ausgewählt, die sowohl in Englisch als auch in Deutsch vorliegen. Dabei bleiben leider relativ wenige übrig.

MCI: Merkmale	
<b>Format</b>	SGML
<b>Quelle</b>	<a href="http://www.elsnet.org/resources/eciCorpus.html">http://www.elsnet.org/resources/eciCorpus.html</a>
<b>Art</b>	längere Texte
<b>Kategorie</b>	Technik, Politik

## NATO

Auf den Webseiten der NATO finden sich einige Pressemitteilungen. Davon sind allerdings nur wenige auch in Deutsch.

NATO: Merkmale	
<b>Format</b>	HTML
<b>Quelle</b>	<a href="http://www.nato.int/docu/pr/">http://www.nato.int/docu/pr/</a>
<b>Art</b>	längere Texte
<b>Kategorie</b>	Politik

Diese Texte waren sowohl relativ einfach zu beschaffen, als auch zu verarbeiten. In der Startphase des Projekts bildeten die NATO-Texte und der MCI-Korpus den Grundstock.

## Verbmobil

Im Rahmen des Verbmobil-Projekts sind eine große Menge an Transkriptionen von Dialogen und Telefongesprächen maschinell nutzbar gemacht worden.

Verbmobil: Merkmale	
<b>Format</b>	ASCII, proprietär
<b>Quelle</b>	am IKW vorhanden
<b>Art</b>	kürzere Texte
<b>Kategorie</b>	Dialog

Geplant war, diese Texte zum Testen des Alignments einzusetzen. Im Projektverlauf hat es sich dann aber so herausgestellt, dass die Texte doch nicht zum Einsatz kamen: Zuerst war der Aligner noch nicht stabil genug zum Testen, später waren wir mit den Ergebnissen zufrieden, so dass wir das Testen mit dem Verbmobil-Korpus nicht mehr für nötig hielten.

### 3.2.2 Auswahl

Den quantitativ größten Anteil machen die Texte aus den Korpora EU und DE-News aus. Die NATO- und MCI-Texte tragen nur relativ wenig bei. Die Bibel-Texte sind aus verschiedenen Gründen (keine echte Übersetzung, verschiedene Struktur) für unsere Zwecke nicht geeignet. Wegen stark unterschiedlichen Aufbaus konnten auch die Texte aus den Linux HOWTOs nicht verwendet werden. Die Texte aus dem Verbmobil-Korpus sollten als Referenz-/Testmaterial für das Alignment dienen, kamen aber bisher nicht zum Einsatz.

## 3.3 Quantitative Angaben

Das KoKS-Korpus besteht im Wesentlichen aus 8 Teilkorpora: DE-News, EU, NATO, MCI, Bibel und Verbmobil sowie den Wörterbüchern „WB“ und „LQL“, die auch in den Korpusbaum integriert sind. Dieser Abschnitt stellt quantitative Daten über die Texte vor.

### 3.3.1 Die Korpora in Zahlen

Tabelle 3.1 zeigt Daten zu den einzelnen Teilkorpora: Anzahl der Dateien und Gesamtgröße (in KB). Insgesamt stehen ca. 26,000 Dateien zur Verfügung bei einer Größe von insgesamt 122 MB. Wie man sieht, ist das EU-Korpus deutlich das größte. Es folgen DE-News, Bibel und Verbmobil. Allerdings wurde (wie bereits erwähnt) das Bibel-Korpus im späteren Projektverlauf nicht mehr benutzt. Auch das Verbmobil-Korpus kam nicht wie ursprünglich geplant zum Einsatz. Der NATO- und das MCI-Korpus spielen bei den quantitativen Angaben nur eine kleine Rolle.

Die reine Größe in KB oder MB sagt noch nicht sehr viel aus. Tabelle 3.2 zeigt daher, aus wieviel Wörtern, Zeilen und Zeichen die einzelnen Teile bestehen. Diese Werte wurden mit dem Unix-Tool `wc` ermittelt, so dass die Angaben nur als Näherung aufzufassen sind. Insbesondere kann die tatsächliche

Korpus	Dateien	Größe [KB]
Bibel	132	10,536
DE-News	2,214	14,542
EU	23,610	93,683
MCI	12	1,782
NATO	62	1,036
Verbmobil	4	3,728
<b>Gesamt</b>	26,034	125,307

Tabelle 3.1: Korpora: Größen (Speicherplatz)

Korpus	Zeilen	Wörter	Zeichen
Bibel	256,200	1,616,120	10,169,096
DE-News	274,959	1,912,206	13,454,497
EU	1,580,780	11,513,213	83,867,545
MCI	56,055	254,391	1,733,279
NATO	7,413	136,803	1,029,366
Verbmobil	44,812	738,267	3,766,074
<b>Gesamt</b>	2,220,219	16,171,000	114,019,857

Tabelle 3.2: Korpora: Wörter und Zeichen

(sinnvolle) Zahl der Wörter davon abweichen. Insgesamt erhält man mit *wc* 16 Mio. Wörter. Diese Anzahl an *tokens* unterscheidet weder Sprache noch werden gleiche Wörter zusammengefasst. Eine genauere Analyse müsste die tatsächliche Zahl an verschiedenen Wörtern (*types*) berücksichtigen.

Schließlich kann man noch die Dateien auswerten, nachdem sie vom Tagger bearbeitet wurden. Tabelle 3.3 zeigt die Datenlage: Hier wurden in den getaggtten Daten (ohne Verbmobil) die Anzahl der Segmente gezählt. Da dieses Zählen vor dem Alignen erfolgt, entspricht diese Zahl in etwa der Anzahl an Sätzen. Die deutschen und englischen Texte bestehen beide aus je etwa 300,000 Segmenten.

Tabelle 3.4 listet die Zahl der Tokens in den deutschen und englischen Texten (ohne Verbmobil) auf. Insgesamt sind es ca. 18.5 Mio. Tokens – davon rund 9 Mio. deutsch und 9.5 Mio. englisch. Wie man sieht, unterscheidet sich diese Zahl etwas von der mit *wc* ermittelten Zahl der Wörter in Tabelle 3.2.

### 3.3.2 Programme zur Auswertung

Die o.a. Daten dienen nur als Näherung, können dafür aber recht unkompliziert gewonnen werden. Grundlage ist die Index-Datei `index.xml`, in der alle Korpus-Dateien verzeichnet sind. Der genaue Aufbau wird in

Korpus	Deutsch	Englisch	Gesamt
Bibel	26,277	24,518	50,795
DE-News	64,558	59,851	124,409
EU	210,690	205,128	415,818
MCI	6,053	5,954	12,007
NATO	2,333	2,386	4,719
<b>Gesamt</b>	309,911	297,837	607,748

Tabelle 3.3: Korpora: Segmente in getaggtten Daten

Korpus	Deutsch	Englisch	Gesamt
Bibel	929,375	1,006,150	1,935,525
MCI	139,498	145,831	285,329
NATO	73,660	75,842	149,502
DE-News	1,112,139	1,261,750	2,373,889
EU	6,724,485	6,997,834	13,722,319
<b>Gesamt</b>	8,979,157	9,487,407	18,466,564

Tabelle 3.4: Korpora: Tokens in getagten Daten

Abschnitt 3.6 vorgestellt, vorerst soll reichen, dass man mit dem Kommando `idx-ls.tcl -t` eine knappe Inhaltsangabe der Datei erhält, nämlich alle Dateinamen. Diese Liste von Dateinamen wird dann abgearbeitet.

Einige Beispiel-Programme zur quantitativen Auswertung befinden sich im Abschnitt C.1.2.

## 3.4 Aufbereitung

### 3.4.1 Allgemeines Vorgehen

Im Folgenden sollen die Grundzüge der Aufbereitung der Korpus-Dateien skizziert werden. Die Aufbereitung umfasst dabei die Schritte, die noch vor der sog. Vorverarbeitung (siehe Abschnitt 5.1 über Normalisierung) durchgeführt werden mussten. Dies sind meistens relativ einfache Aufgaben wie Verzeichnisstrukturen anlegen, Dateinamen vereinheitlichen, aber auch kniffligere Probleme wie z.B. feststellen, ob die Dateien den Inhalt enthalten, den sie vorgeben. Wegen der großen Zahl der Dokumente können diese Schritte schlecht manuell durchgeführt werden, sondern es bietet sich an, Skripte zu schreiben, die diese Aufgaben übernehmen.

#### Fallstudie: EU

Die Beschaffung der einzelnen Teil-Korpora war unterschiedlich aufwändig. Exemplarisch sollen hier einige Schritte vorgestellt werden, die zum Aufbau des EU-Korpus durchgeführt wurden:

**Dateien holen** Die Texte (Veröffentlichungen der Pressestelle u.ä.) sind nicht als Paket erhältlich, sondern wurden einzeln mit dem Programm `wget` aus dem WWW geholt. Dazu wurde eine Liste mit URLs erstellt, die dann der Reihe nach (mit kleinen Pausen) abgearbeitet wurde. Da die Dokumente nur über eine EU-Suchmaschine zugänglich sind, mussten die Aufträge an diese Suchmaschine simuliert werden. Das hatte allerdings auch zur Folge, dass nicht alle Anfragen korrekt bearbeitet wurden (s.u.). Schließlich wurden die Dateien in einfachere Namen umbenannt.

**Fehlende Dateien finden** Nicht alle Dokumente liegen in mehreren Sprachen (insb. Deutsch und Englisch) vor. Außerdem werden nicht alle Anfragen erfolgreich bearbeitet. Also mussten die Log-Dateien ausgewertet werden, ob alle angeforderten Dateien auch lokal vorhanden sind. Somit konnten fehlende Dateien entdeckt und nachträglich angefordert werden, so dass jedes Dokumentenpaar vollständig war. Dies erfolgte hauptsächlich mit UNIX-Tools und einfachen Shell-Skripten.

**Dateien mit falscher Sprache entdecken** Einige Dokumente waren hinsichtlich ihrer Sprache falsch ausgewiesen. Jedes EU-Dokument beginnt mit einem einigermaßen einheitlichen Vorspann, indem alle Sprachen, in denen das Dokument vorliegt, aufgelistet werden. Tauchen in dieser Liste nicht Englisch und Deutsch auf, so ist das Dokument für unsere Zwecke uninteressant – und offensichtlich falsch klassifiziert.

So kam es bspw. häufiger vor, dass sich hinter einer Datei, die dem Namen nach deutschen Inhalt vorzuweisen hätte, ein französischer oder englischer Text verbarg. Mit UNIX-Tools wie `grep` und `cut` konnten die meisten dieser Fälle jedoch entdeckt werden.

**Interessanten Text ausschneiden** Für die weitere Verarbeitung war es sinnvoll, einigen Vor- und Nachspann aus den Dokumenten zu löschen, da sie für den Übersetzungskontext irrelevant schienen. So ist oben erwähnte Auflistung der vorhandenen Sprachen eines Dokuments für die weitere Verarbeitung uninteressant. Außerdem enthielten einige Dokumente falsche HTML-Tags. Auch diese Aufgabe war mit einfachen Shell-Skripten zu bewältigen.

**In Textformat konvertieren** Schließlich wurden die Dateien in Text-Format konvertiert (`lynx -dump -nolist`). Zwar ermöglichen die Skripte aus dem Normalisierungsmodul (s. D.1) auch eine HTML-Konvertierung, in diesem Fall war aber noch Nachbearbeitung notwendig.

**Unnützen Vorspann löschen** Zu diesem Zeitpunkt befindet sich der Text fast in brauchbarer Form. Es wurde aber noch ein weiterer Filter angewandt, der in allen Dokumenten vorkommenden Vorspann löscht (wie z.B. eine Datums-Zeile).

**Manuelle Nachbearbeitung** Im Idealfall konnten alle diese Schritte automatisch durchgeführt werden. Einige der Dokumente wiesen aber eine abweichende Struktur auf, so dass diese Sonderfälle manuell nachbearbeitet wurden. Untersucht wurden insgesamt ca. 13500 Dateien, dabei war die Zahl der Ausreißer jedoch erträglich gering.

**Index erstellen** Als letzter Schritt wurde für jeden Jahrgang eine `index.xml` Datei erstellt. Die Index-Datei enthält alle wichtigen Informationen über die Dateien in einem Korpus, siehe dazu Abschnitt 3.6.1.

### 3.4.2 Spezielle Probleme

Fast immer verlief diese Aufbereitung einigermaßen problemlos. In manchen Fällen kam es aber zu Problemen, von denen zwei hier erwähnt werden sollen.

#### EU: Sprachwirrwarr

Bei der Aufbereitung der Texte aus dem EU-Korpus stellte sich heraus, dass einige Dokumente nicht in der Sprache verfasst waren, für die sie ausgewiesen waren. Der Grund dafür liegt zum einen in der Konfiguration des Webservers, der die EU-Dokumente bereitstellt, und zum anderen darin, dass Dokumente von der jeweiligen Veröffentlichungsstelle falsch klassifiziert worden sind.

Das erste Problem zeigte sich dadurch, dass der Webserver bei Anfragen nach einem Dokument in einer Sprache, in der das Dokument nicht vorliegt, keinen Fehler meldete, sondern einfach eine englische Fassung schickte. Das zweite Problem führte dazu, dass ein Dokument z.B. als französisch ausgewiesen wurde, in Wirklichkeit aber deutsch war. Ziel der Aufbereitung war es, nur Dateien mit kennzeichnenden Namen zu erhalten, so bezeichnet `1446-de.txt` das deutsche und `1446-en.txt` das entsprechende englische Dokument. Durch Vorarbeit (s.o.) konnten alle „falschen“ Dokumente (kein dt.-engl. Paar, vom Server falsch geliefert etc.) gelöscht werden, so dass als letzte Aufgabe blieb, die Dokumente zu finden, die in ihrer Sprache falsch klassifiziert worden waren.

Um die Sprache eines Dokuments festzustellen, konnten wir uns nicht auf Angaben innerhalb der Datei (wie `<meta>` Tags verlassen). Stattdessen wurde die Identifikation mit Hilfe von n-Grammen durchgeführt (Sibun und Reynar, 1996). Mit dieser einfachen, aber dennoch effizienten Technik konnten „schwarze Schafe“ entdeckt werden. Zum genauen Vorgehen siehe Abschnitt 3.5.

#### DE-News: falsche Zuordnung

Ein anderes Problem trat bei den Texten im DE-News-Korpus auf. Dort sind die einzelnen Nachrichtentexte durch eine vorangestellte ID gekennzeichnet:

```
<DOC de-news-1996-10-29-1>
<H1>
Unterschiedliche Reaktionen auf das Herbstgutachten
</H1>
Das Herbstgutachten der fuehrenden Wirtschaftsforschungsinstitute
ist auf unterschiedliche Reaktionen gestossen...
```

Im Idealfall entspricht die ID im deutschen Text der im englischen Text. Es kommt aber auch vor, dass zwei nicht zusammengehörige Texte die gleiche ID haben oder dass ein Text nur in einer Sprache vorliegt (in der korrespondierenden Datei fehlt ein Text mit passender ID). Die Fälle, in denen das Paar unvollständig ist, lassen sich leicht aufspüren, ebenso machen auch die Dateien wenig Probleme, in denen die Ordnung der einzelnen Nachrichten-Texte in beiden Sprachen unterschiedlich ist. Um den Fall „gleiche ID, falscher Text“ zu lösen, benutzen wir unsere Alignment-Tools, um die Überschriften der einzelnen Blöcke zu alignen.

Vorgehen:

1. Überschriften aus den Texten extrahieren
2. Überschriften taggen
3. Abstandsmatrizen mit WB2 berechnen (s. Abschnitt 7.3.1)
4. Best-Cell-Alignment finden (s. Abschnitt 7.4.2)
5. Überkreuzungen zählen und auswerten

Die Hypothese dabei ist, dass Dateien, in denen viele Überkreuzungen beim Alignment entstehen, vermutlich falsche IDs tragen. Die Überprüfung der Ergebnisse bestätigte diese Hypothese:

Überkreuzungen	Häufigkeit
2	212
3	65
4	45
5	20
6	12
7	3
8	2
10	1
14	1

Die Spalte „Überkreuzungen“ gibt die Zahl der überkreuzten IDs in einem deutsch-englischen Textpaar an. Je mehr Überkreuzungen auftauchen, umso wahrscheinlicher ist es, dass die Überschriften und IDs in dem deutschen Dokument nicht zu dem englischen Dokument passen. Die Spalte „Häufigkeit“ zeigt, dass es in den meisten Fällen nur wenig Überkreuzungen gab.

Davon „wirklich“ überkreuzt:

- der 14er
- im 10er jeweils ein Block en/de ohne Pendant
- bei den 6ern drei Datei-Pärchen nicht passend
- bei den 5ern zwei Datei-Pärchen nicht passend
- der Rest ( $\leq 4$ ) vermutlich ok

Insgesamt waren also wenige Nachrichten-Texte mit falschen IDs versehen. Unser Ansatz zeigt aber, dass auch diese wenigen Ausreißer fast automatisch gefunden werden konnten.

## 3.5 Sprachidentifikation

Wie bereits in Abschnitt 3.4 erwähnt, war es in manchen Fällen notwendig, die Sprache eines Dokuments zu bestimmen. Wir haben dazu eine Reihe von kleinen Programmen entwickelt, die diese Aufgabe weitgehend automatisch erledigen. In diesem Abschnitt sollen die zu Grunde liegende Theorie und ein Anwendungsbeispiel vorgestellt werden.

### 3.5.1 Mathematischer Hintergrund

Es gibt viele Möglichkeiten, automatische Sprachidentifikation durchzuführen (Dunning, 1994). Eine relativ einfache, aber dennoch effiziente Variante ist die n-Gramm Methode (Sibun und Reynar, 1996). Dazu werden zuerst die in einem Text auftretenden n-Gramme gezählt. Diese Zähl-Statistik kann man vereinfacht als Wahrscheinlichkeitsverteilung auffassen. Die Idee ist jetzt, die Verteilung eines unbekanntes Dokuments (Testdokument) mit den Verteilungen von Dokumenten, deren Sprache bekannt sind (Trainingsdokumente), zu vergleichen. Im günstigsten Fall ergibt sich daraus ein Bild, zu welcher bekannten Verteilung die des Testdokuments am ähnlichsten ist.

#### Relative Entropie

Den Vergleich von zwei Wahrscheinlichkeitsverteilungen  $p(x)$  und  $q(x)$  kann man mit Hilfe der *relativen Entropie* durchführen. Die relative Entropie (auch bekannt als *Kullback-Leibler Abstand* oder *I-divergence*) zwischen zwei Verteilungen spiegelt die Menge an zusätzlicher Information wieder, die nötig ist, um die zweite Verteilung darzustellen, wenn man die erste Verteilung und einen optimalen Code benutzt. Mit anderen Worten: Man kann relative Entropie als ein Maß der Ähnlichkeit von zwei Verteilungen auffassen. Allerdings ist sie kein Abstandsmaß, weil sie nicht symmetrisch ist.

Seien  $p$  und  $q$  Wahrscheinlichkeitsverteilungen, so ist die relative Entropie  $D$  definiert als:

$$D(p \parallel q) := \sum_{x \in \mathcal{X}} p(x) \cdot \log \frac{p(x)}{q(x)} \quad (3.1)$$

Dabei ist  $\mathcal{X}$  die Menge aller Ereignisse (z.B. n-Gramme, s.u.). Der Wert von  $D$  liegt zwischen 0 und  $\infty$  ( $0 \cdot \log \frac{0}{q} := 0$  und  $p \cdot \log \frac{p}{0} := \infty$ ).

#### Anwendung

Die Anwendung besteht aus einer Trainings- und einer Testphase. Dazu wird aus jedem Korpus (Dokument) von bekannter Sprache ein Teil ausgewählt. Für jede Sprache, die man später identifizieren möchte, erhält man so eine Trainingsmenge. Für jede dieser Trainingsmengen werden folgende Schritte durchgeführt:

1. n-Gramme zählen
2. Werte ausgleichen (*smoothing*)
3. Werte in Wahrscheinlichkeiten umrechnen

Smoothing sorgt dafür, dass Ereignisse (n-Gramme), die zwar in der Test-, nicht aber in der Trainingsmenge auftauchen, nicht den Entropie-Wert auf  $\infty$  bringen. Daher wird für jedes Ereignis, das nicht in einer bestimmten Sprache auftaucht, welches aber in einer der anderen vorkommt, der Wert 0.5 zur Häufigkeit addiert. Ein Ereignis in der Testmenge, das in keiner der Trainingsmengen auftaucht, wird ignoriert. Man könnte dieses Smoothing durch anspruchsvollere Techniken ersetzen (Manning und Schütze, 1999, ch. 6), die von uns verwendeten einfachen Methoden haben sich aber als hinreichend erwiesen.

Für die Testphase wählt man einen kleinen Teil aus der Testmenge aus. In der Regel ist es nicht notwendig, die Verteilung über der ganzen Menge zu berechnen, meistens genügt ein Ausschnitt, was die ganze Identifikations-Prozedur auch beschleunigt. Für diese Menge erzeugt man eine Verteilung wie oben beschrieben, allerdings ohne den Ausgleichsschritt. Dann werden die Entropie-Werte zwischen der Testmenge und allen Trainingsmengen berechnet.

Die Testmenge wird mit der Sprache identifiziert, für die der zugehörige Entropie-Wert minimal wird. Mit anderen Worten: Sei  $p$  die Verteilung, die zur Testmenge  $P$  gehört und seien  $q_i$  die Verteilungen der

Trainingsmengen  $Q_i$  mit den jeweiligen Sprachen  $L(Q_i)$ . Ferner seien  $n$  Trainingsmengen gegeben ( $1 \leq i \leq n$ ). Dann setzt man  $L(P) = L(Q_x)$ , wobei

$$x = \arg \min_{1 \leq i \leq n} D(p \parallel q_i) \quad (3.2)$$

### 3.5.2 Beispiel

N-Gramme kann man mit dem Programm `ngram.rb` erstellen, für eine genauere Beschreibung siehe Abschnitt C.1.4.

```
koks@nunix:~/artus$ echo "Hallo Welt" | ./ngram.rb -n
we:1
all:1
elt:1
hal:1
llo:1
lo :1
lt :1
o w:1
wel:1
```

Für die Identifikation von Dokumenten gibt es die Programme `cfile.rb`, `cclient.rb` und `cserver.rb`. Ersteres kann eingesetzt werden, wenn nur eine kleine Menge von Dokumenten analysiert werden soll. Bei großen Aufträgen bietet sich die Client/Server-Version an, weil dann die Trainingsdaten nicht für jeden Aufruf neu geladen werden müssen. Alle diese Programme werden in Abschnitt C.1.4 vorgestellt.

Abschließend sei noch darauf hingewiesen, dass n-Gramme auch beim Alignment-Modul eingesetzt werden, s. dazu Abschnitt 7.3.1.

## 3.6 Verwaltung und Organisation

### 3.6.1 XML

Die Korpus-Dateien wurden bis zu ihrem Eintrag in die Datenbank in einem eigenen Korpus-Verzeichnis abgelegt. Bei Dokumenten, die aus dem WWW gesammelt wurden, wurde teilweise die vorgefundene Verzeichnisstruktur übernommen. Ansonsten wurden große Dateisammlungen logisch in Teile unterteilt. Informationen zu den einzelnen Dokumenten wie Sprache, Herkunft, Format und die Zuordnung der englischen und deutschen Teile zueinander wurde in einer Index-Datei abgelegt. Außerdem enthält ein Meta-Index Angaben über die Position der einzelnen Index-Dateien im Korpus-Verzeichnis. So sind alle Informationen über die Dokumente von einer zentralen Stelle aus auffindbar.

#### Motivation

Die Informationen über Dokumente werden in XML-Dateien abgelegt. Als Gründe für den Einsatz von XML sprechen:

- einfache Möglichkeit, strukturierte Daten zu speichern
- Index-Dateien können leicht von Hand erstellt/gewartet werden
- XML-Tools und Libraries für alle vom Projekt eingesetzten Programmiersprachen ermöglichen automatische Erstellung/Auswertung
- leichte Weiterverwendung in anderen Projekten, die nur das Korpus benutzen wollen

Das KoKS-Korpus besteht aus mehreren Teil-Korpora (s. Abschnitt 3.2). Die Informationen über die einzelnen Teilkorpora werden in der Datei `index.xml` abgelegt. Diese Index-Datei enthält Informationen über die einzelnen Dateien, Sprache, Herkunft etc. Somit ist es möglich, durch Auswerten der `index.xml` Datei gezielt Dateien im Korpus-Baum zu finden, die spezielle Kriterien erfüllen.

**Beispiel**

Die Index-Dateien können manuell oder, was viel einfacher ist, mit dem Programm `idx-add.tcl` erstellt werden. Eine detaillierte Beschreibung zu diesem Programm befindet sich in Abschnitt I.3.4. Als Beispiel soll eine einfache Index-Datei erstellt werden, die Verweise auf eine englische (`e.ps`) und eine deutsche (`d.txt`) Datei enthält:

```
koks@nunix:/tmp > idx-add.tcl -p -l de d.txt plain -p -l en e.ps ps
```

Das Dokument besteht aus zwei Teilen, die deutsche Fassung liegt als ASCII-Datei vor, die englische im Postscript-Format. Man erhält folgende Datei (Ausschnitt):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE mapping SYSTEM "/home/koks/share/mapping.dtd">
<mapping>
<document>
<part lang="de">
<file>
<filename format="plain">d.txt</filename>
</file>
</part>
<part lang="en">
<file>
<filename format="ps">e.ps</filename>
</file>
</part>
<info>
...
</info>
</document>
</mapping>
```

Der Info-Block bleibt leer und kann noch mit Leben gefüllt werden. Die so erstellte Index-Datei kann man sich mit `cat` ansehen oder in formatierter Darstellung mit `idx-ls.tcl`. Im einfachsten Fall gibt `idx-ls.tcl` den Inhalt der `index.xml` Datei im aktuellen Verzeichnis aus (so vorhanden), der Schalter `-f` gibt zusätzlich das Format der Dateien an:

```
koks@nunix:/tmp$ > ls
d.txt e.ps index.xml
koks@nunix:/tmp$ > idx-ls.tcl
de: d.txt
en: e.ps
koks@nunix:/tmp$ > idx-ls.tcl -f
de: (plain) d.txt
en: (ps) e.ps
```

**DTD**

Die Struktur von `index.xml` gehorcht den Regeln, die in der Document Type Definition `mapping.dtd` aufgestellt sind. Erstellt man die Index-Dateien von Hand, muss man diese DTD beachten. Eine ausführliche Beschreibung der DTD steht in Abschnitt C.2.1.

**3.6.2 Korpus-Browser**

Zur einfacheren Navigation durch den Korpus-Baum gibt es einen Korpus-Browser. Mit diesem kann man sich `index.xml` Dateien ansehen und sich die darin aufgelisteten Korpus-Dateien in den verschiedenen Verarbeitungsstufen (Original, getagged, aligned) anzeigen lassen.

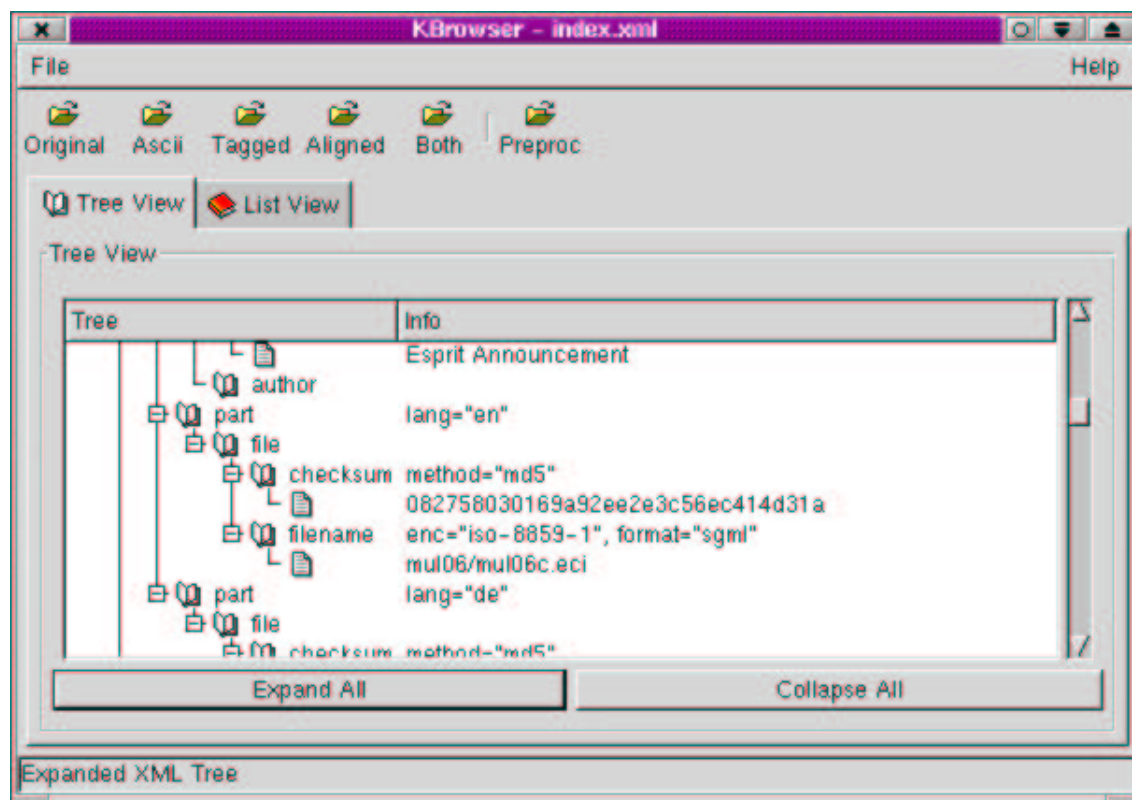


Abbildung 3.1: KBrowser – Baum-Ansicht

Der Korpus-Browser hat zwei Darstellungsarten: Eine Baum-Ansicht (Explorer), in der alle Knoten des XML-Dokuments expandierbar angezeigt werden, und eine Listen-Ansicht, die nur die Dateinamen mit dem dazugehörigen Format zeigt. Abbildung 3.1 zeigt den KBrowser in der Baum-Ansicht. Markiert man einen Knoten im Baum, der einen Dateinamen repräsentiert, so kann man über Buttons im Toolbar verschiedene Funktionen auswählen:

**Original** zeigt das Dokument in Rohfassung

**ASCII** zeigt das Dokument in ASCII

**Tagged** zeigt den Text mit POS-Tags

**Aligned** zeigt den Text mit Alignment-Markierungen

**Both** zeigt deutschen und englischen Text parallel

Die letzte Funktion funktioniert dabei nur im speziellen Fall, wenn jede Sprachvariante eines Dokuments nur aus einer Datei besteht – die DTD erlaubt zwar auch kompliziertere Fälle, im KoKS-Projekt tauchten aber keine auf.

Zusätzlich hat man die Möglichkeit, das `preproc.py` Skript zu starten. Dazu markiert man einen Dokument-Knoten. Abbildung 3.2 zeigt die zur Verfügung stehenden Operationen:

**Convert** Konvertiert Dokumente ins ASCII-Format

**Tag** Startet den Tagger

**Align** Startet das Aligment

**Enter DB** Trägt Dokumente in die Datenbank ein

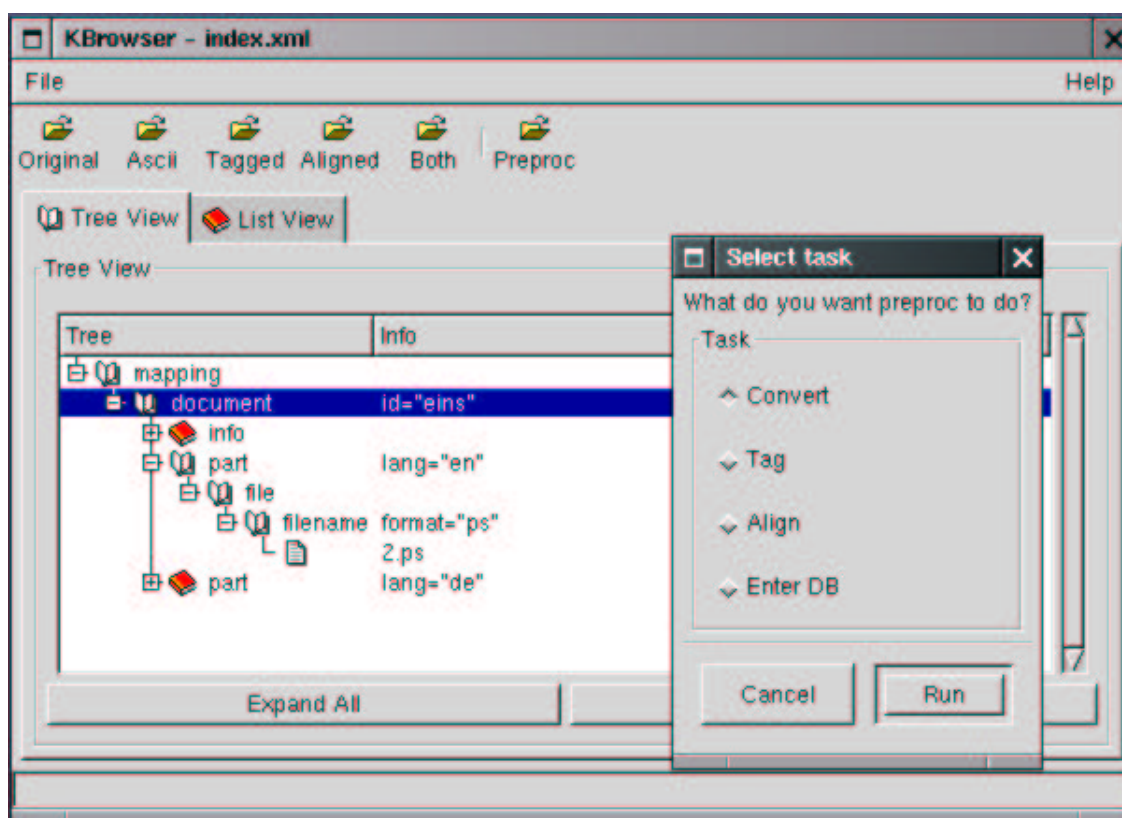


Abbildung 3.2: KBrowser in Aktion

Zu den Möglichkeiten, die das Normalisierungs-Programm `preproc.py` bereit stellt, siehe auch Abschnitt 5.1.

Ist man hauptsächlich an den Dateien ohne die ganze Zusatzinformation interessiert, kann man in die Listen-Ansicht wechseln (s. Abbildung 3.3). Auch hier lassen sich die Dokumente in den verschiedenen Verarbeitungsstufen anzeigen.

In der jetzigen Version reicht der Browser zum Navigieren durch die Korpus-Dateien aus. Mögliche Erweiterungen wären eine Suchfunktion, mit der nach Dokumenten mit bestimmten Eigenschaften gesucht werden kann, und eine Möglichkeit zum (Nach-)Bearbeiten der XML-Datei.

### 3.7 Ausblick und Verbesserungen

Die verwendeten Texte stammen größtenteils aus dem gleichen Bereich (Nachrichten, Pressemitteilungen) bzw. aus ähnlichen Kategorien (Politik). Eine größere Streuung an Texttypen und Genera scheint erstrebenswert.

Die Struktur der Dokumente könnte besser analysiert werden und diese Ergebnisse für das Alignment nutzbar gemacht werden. Derzeit erfolgt eine Textnormalisierung, die alle Textblöcke (Überschriften, Absätze, Tabellen) gleich behandelt.

Bei den EU-Texten tauchen häufig Tabellen auf. Deren Inhalt wird beim Konvertieren und Normalisieren meistens in unbrauchbaren Textmüll verwandelt. Da der Inhalt von Tabellen für die Kollokationsuche vermutlich wenig Interessantes enthält, könnte man ihn bspw. ganz weglassen. Nicht immer sind Tabellen allerdings leicht als solche erkennbar. Bei HTML Dokumenten kann man sich an `<table>` Tags orientieren, bei reinen ASCII Texten ist eine Identifikation schwieriger (z.B. durch viel Whitespace, Striche). Andererseits dienen `<table>` Tags in HTML auch häufig zur Formatierung, so dass man irrtümlicherweise den

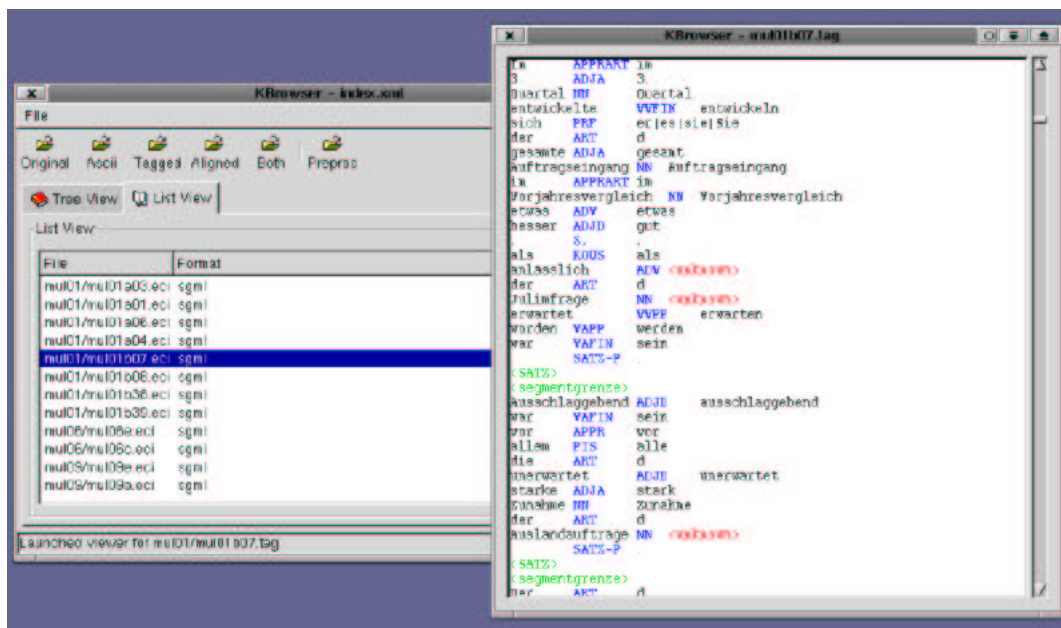


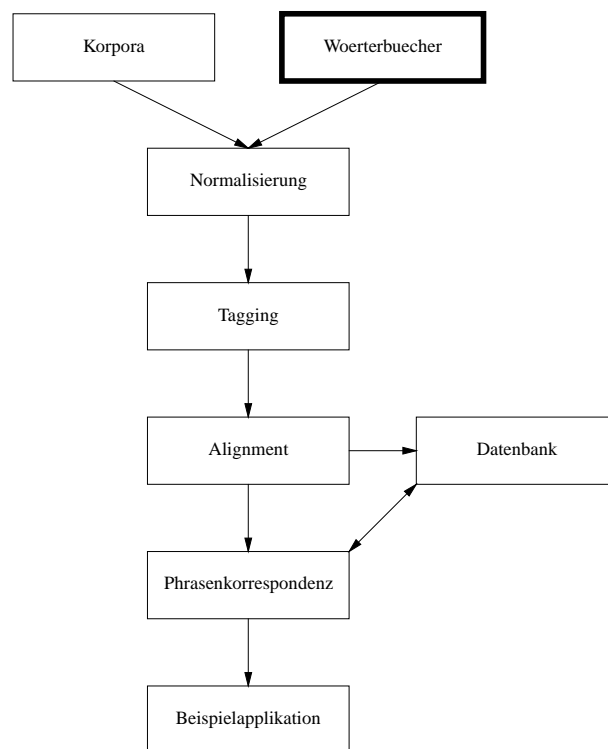
Abbildung 3.3: KBrowser – Listen-Ansicht

interessanten/wichtigen Text als „unwichtig“ abtut und verwirft.

Die Konvertierung von Binär-Formaten (PS, PDF, Word) kann durch bereits vorhandene Programme erfolgen, die Ausgabe ist allerdings nicht immer wirklich zu gebrauchen. Problematisch sind hier häufig mehrspaltiger Text, Tabellen, Bildunterschriften.

# Kapitel 4

## Wörterbücher



### Inhaltsverzeichnis

---

4.1	Einleitung/Motivation . . . . .	30
4.2	Dictionary Entry Parser . . . . .	30
4.3	Aufbau der Textformate . . . . .	31
4.4	Zusammenfassung . . . . .	34
4.5	Ausblick . . . . .	35

---

## 4.1 Einleitung/Motivation

Wörterbücher sind für das KoKS-System ein essentiell wichtiger Bestandteil. Deshalb wurden sie auch schon früh im Projektverlauf für unser System verfügbar gemacht. Das System verwendet nur zweisprachige Wörterbücher. Einsprachige Wörterbücher, welche Kollokationen einem Einzelwort zuordnen, könnten später eingebracht und so zusätzlich zu mehr Verständlichkeit der Kollokationen beitragen. Einige Wörterbücher enthielten Mehrworteinträge. Diese werden vom KoKS-System als initiale Kollokations-Prototypen genutzt. Wir verstehen unter dem Begriff Kollokations-Prototypen Multiwortkonstruktionen, welche schon von Beginn an dem System bekannt waren, also aus den Wörterbüchern stammen (man könnte sie auch Initial- oder Ausgangskollokationen nennen). Später wird das System eigene Kollokations-Kandidaten anhand dieser Kollokations-Prototypen (s. Abschnitt 8.1) finden.

Die Wörterbücher werden aber auch zum Alignment eingesetzt (der im Projekt entstandenen Aligner "Mavis" (s. Abschnitt 7.3.1) verwendet sie als Abstandsmaß (WB2)).

Die im Projekt eingesetzten Wörterbücher kommen aus verschiedenen Quellen. Wir durchsuchten dafür das Internet und fanden dabei einige brauchbare Wörterbücher:

- Ding (TU-Chemnitz) z.T. mit Kollokationseinträgen, mit ca. 150000 Einträgen<sup>1</sup>

Beispiel: Kopie [f]; Abbild [n] :: copy

Aufbau: Links neben dem Sprachentrenner (Delimiter) ':::' stehen zwei Übersetzungen in Deutsch welche durch ';' getrennt sind, jeweils gefolgt von einem Wortklassen-Tag, der in eckigen Klammern eingeschlossen ist. Rechts folgt der englische Eintrag.

Dateien: Dieses Wörterbuch besteht nur aus einer Datei mit jeweils einer oder mehreren Übersetzungen pro Sprache in einer Zeile.

- The Internet Dictionary Project/Tyler Chambers mit ca. 9700 Einträgen<sup>2</sup>

Beispiel: abdicator\*Abdanker, Verzichter, Thronverzichter[Noun]

Aufbau: Die Sprachen werden durch 'Tabulator' (hier ersetzt durch \*) getrennt, links steht der englische Eintrag, rechts durch ',' getrennt stehen drei Übersetzungsvarianten für Deutsch, eine enthält einen Wortklassen-Tag in eckigen Klammern.

Dateien: Es existiert jeweils eine Datei für Deutsch-Englisch und Englisch-Deutsch mit auf der rechten Seite stehenden Übersetzungen des linken Eintrags.

Außerdem wurden in unserem Institut vorhandene Wörterbücher eingesetzt:

- Das LQL-Wörterbuch mit 115324 Einträgen pro Sprache, diese enthalten ca. 36000/42000 Kollokationseinträge<sup>3</sup>
- Die Wörterbücher MDI und Collins wurden bearbeitet, hatten aber zu viele Fehler. Deshalb stellten wir sie zurück.

## 4.2 Dictionary Entry Parser

Da die Wörterbücher alle in verschiedenen Formaten (s. Aufbau der Textformate 4.3) vorlagen, wurde für jedes Wörterbuch ein eigener Dictionary Entry Parser (DEP) geschrieben.

Ein DEP ist ein Programm, welches die Lexikoneinträge aus einem Wörterbuch parst, in ein für unser System einlesbares Format umformt und verschiedene verfügbare Zusatzinformationen (z.B. Tags) extrahiert.

Die DEP-Programmierung erfolgte z.T. in VisualBasic (s. Abschnitt J.1.1) (für die einfachen Formate, s. Aufbau der Textformate 4.3) oder in Ruby s. Abschnitt J.1.2) (LQL).

<sup>1</sup><http://www.tu-chemnitz.de/dict>

<sup>2</sup><http://www.june29.com/IDP/>

<sup>3</sup>Aufbau/Beispiel: (s. Aufbau der Textformate 4.3) bzw. siehe <http://www.cl-ki.uni-osnabrueck.de/~ulf/uni/ws98-99/lexikon/>

In allen Wörterbüchern waren Fehler, sowohl semantische (falsche Übersetzungen, fehlerhafte Morphologie) als auch strukturelle (fehlende Klammern, fehlende Delimiter, englisch und deutsche Einträge durcheinander; bzw. nicht sauber getrennt). Diese wurden z.T. automatisch, oft auch per Hand korrigiert, manche waren auch irreparabel. An der Form der aufgetretenen Fehler konnten wir erkennen, dass die Wörterbuchdateien ursprünglich wahrscheinlich nur mit dem Unix-Kommando 'grep'-ähnlichen Programmen durchsucht wurden, um Einträge zu finden, bzw. dass sie nicht für maschinelle Verarbeitung vorgesehen waren oder von Hand erstellt wurden. Andernfalls hätten solche, wie im Beispiel angeführten, Fehler nicht auftreten dürfen.

Beispiel für einen Klammerfehler:

```
Kopie [f; Abbild [n] :: copy
```

Beispiel für einen ausgelassenen Delimiter (zwischen 'Kopie' und 'Abbild'):

```
Kopie Abbild :: copy
```

Hier würde ein völlig falscher Eintrag in unserer Datenbank erzeugt, welcher 'Kopie Abbild' auf 'copy' mappert (und nicht Kopie:copy, Abbild:copy): dieser Fehler kann nur von Hand ausgemerzt werden, da er nicht automatisch erkannt wird.

## 4.3 Aufbau der Textformate

### 4.3.1 Einfaches Format

Die meisten Wörterbücher hatten folgende einfache Form (in drei Gruppen unterteilt):

#### Form 1/3

Sprache-A-1.Eintrag "Delimiter" Sprache-B-1.Eintrag

Beispiel: Auto\*car

Wobei: Sprache-A-1.Eintrag: Auto.

Wobei: "Delimiter": \*

Wobei: Sprache-B-1.Eintrag: car

Hierbei wurde nach dem Delimiter gesucht und dort nach Sprachen getrennt, eine Zeile in die Sprache-A-Datei, sowie eine Zeile in die Sprache-B-Datei geschrieben. Es wurden pro Wörterbuch drei Ausgabedateien erzeugt (außer LQL). Die erste enthielt den Wörterbucheintrag in der Form:

Sprache-A-Eintrag "Delimiter" Sprache-B-Eintrag konkateniert mit dem Originaleintrag  
(im oben aufgeführten Beispiel: Auto\*car)

Die komplette Ausgabezeile sieht wie folgt aus:

```
Auto::car\#\#Auto*car
```

Der Originaleintrag wurde mit ausgegeben, weil er in manchen Fällen zusätzliche Informationen (Tags, morphologische Informationen) enthielt, die später noch ausgewertet werden sollten.

Diese Datei (im folgenden Text '1. Datei' genannt) enthält also die Informationen für beide Sprachen und den Originaleintrag. Im Unterschied dazu wurden zwei weitere Dateien (im folgenden Text '2./3. Datei' genannt) erzeugt, die jeweils nur den Lexikoneintrag pro Zielsprache enthalten. Beide Dateien haben gleich viele Zeilen, jede Zeile der einen hat somit einen eindeutigen "Partner" in der anderen Datei. Jede Zeile endet mit einem speziellen XML-Tag: `¡KOKS¿`. `¡ABSATZ¿¡/KOKS¿`. Dieses Format wurde gewählt, um die Wörterbucheinträge durch den Tagger analysieren zu können, da dieser nur einsprachige Dateien verarbeitet und Satzende- und Absatzmarken braucht.

Datei Sprache A - Datei 2 (hier Deutsch):

```
Auto<KOKS>. <ABSATZ></KOKS>
```

Datei Sprache B - Datei 3 (hier Englisch):

```
car<KOKS>. <ABSATZ></KOKS>
```

**Form 2/3**

Sprache-A-1.Eintrag "Delimiter1" Sprache-B-1.Eintrag "Delimiter2" Sprache-B-2.Eintrag

Beispiel: Auto\*car;limo

Hierbei trennt Delimiter1 die Sprachen und Delimiter2 zwischen zwei Übersetzungsvarianten (in einer Sprache, hier Englisch). Für jede Übersetzungsvariante wurde eine Zeile in '1.Datei' und '2./3. Datei' erzeugt, welche in diesem Beispiel in der Sprache A zweimal den gleichen Eintrag enthält und dazu in Sprache B die Übersetzungsvarianten.

Die erzeugte Ausgabe in '1. Datei' sieht wie folgt aus:

```
Auto::car\#\#Auto*car;limo
Auto::limo\#\#Auto*car;limo
```

Die erzeugte Ausgabe in '2./3. Datei' sieht wie folgt aus:

```
Datei Sprache A (hier Deutsch):
Auto<KOKS>. <ABSATZ></KOKS>
Auto<KOKS>. <ABSATZ></KOKS>
```

```
Datei Sprache B (hier Englisch):
car<KOKS>. <ABSATZ></KOKS>
limo<KOKS>. <ABSATZ></KOKS>
```

**Form 3/3**

Sprache-A-1.Eintrag "[Kennzeichen]" "Delimiter1" Sprache-B-1.Eintrag "Delimiter2" Sprache-B-2.Eintrag

Beispiel: Auto[s]\*car;limo

Wobei [s] für sächlich steht.

Beispiel: Abbruch [m] (Sport) :: break-off; stop; stopping

Wobei [m] für männlich steht und (Sport) den Bedeutungskontext kennzeichnet.

Ausdrücke der Form [Kennzeichen] kamen in verschiedensten Ausprägungen vor, konnten aber nur wenig genutzt werden, da sie nicht kontinuierlich vorhanden waren. Sie wurden nur im LQL-Wörterbuch ausgewertet.

**4.3.2 Das LQL Wörterbuch**

Das LQL Wörterbuch hatte eine wesentlich kompliziertere, aber fast maschinenlesbare Form:

Sprache-A-Eintrag "Delimiter" "Klammerstruktur"

Beispiel: Abbruch ^ ( ... )

```
Wobei: Sprache-A-Eintrag: Abbruch
Wobei: 'Delimiter': ^
Wobei: 'Klammerstruktur': ( ... )
```

Die "Klammerstruktur" ist LISP-Code-ähnlich. Es wurden aus einer Fülle von Informationen die für uns wichtigen herausgezogen. Außerdem traten strukturelle Fehler zu Tage. Diese wurden mit mehreren Korrektur- und Filterläufen und auch per Hand repariert. Die Fehler waren zum einen fehlende Klammern, zum anderen defekte Schlüsselwörter, welche u.a. willkürlich getrennt waren.

Beispiel für einen defektes Schlüsselwort:

```
collocat( source: auf ~ verkaufen targ( tar
get( word: to sell for demolition))
```

Hier ist das Schlüsselwort 'target' falsch getrennt.

Solche Fehler wurden z.T. mit Hilfe der mächtigen Ersetzungsfunktion des UNIX-Editors Vi behoben, danach waren mehrere Parse-Durchläufe mit jeweils neu angepaßten Scripten (s. Abschnitt J.1.2) notwendig, um Schritt für Schritt die gewünschte Schnittstellenstruktur herzustellen. Das LQL-Wörterbuch hatte die Besonderheit, kontinuierlich Wortklasseninformation (eine Art Tag) zu enthalten, welche in ein eigenes Tagset (s. Abschnitt E.3.4) (ähnlich dem IMS-Tagset) eingeflossen sind. Diese Informationen waren natürlich sicherer als vom Tagger generierte Tags.

Wir erzeugten vier Dateien, je eine für Einzelwörter pro Sprache und je eine für Kollokationen pro Sprache.

Beispiel für einen etwas komplizierteren, alles enthaltenden Eintrag:

```
Abbruch ^ ( pronunc: A.bbruch hom
( pos: n gender: m morph: no pl sens( sensnum: a tran( word: demolition)
tran( word: pulling down)
collocat( source: auf ~ verkaufen targ( target( word: to sell for demolition)))
collocat( source: auf ~ stehen targ( target( phrase: to be scheduled
.or. due for demolition, to be condemned))))
sens( sensnum: b tran( word: breaking off)
tran( word: stopping)
collocat( source: einem Land mit ~ der diplomatischen
Beziehungen drohen targ( target( word: to threaten to
break off diplomatic relations with a country)))
collocat( source: es kam zum ~ des Kampfes targ( target(
word: the fight had to be stopped))))
sens( sensnum: c tran( word: harm word: damage)
collocat( source: einer Sache explain: dat targ( target(
phrase: ~ tun to harm .or. damage sth, to do (some) harm .or. damage to sth)))
collocat( source: das tut der Liebe keinen ~ targ( target(
word: it doesn't harm their/our relationship)))
collocat( source: das tut unseren Interessen ~ targ( target(
word: that is detrimental to our interests)))
collocat( source: das hat der Fröhlichkeit keinen ~ getan targ(
target( word: it didn't spoil the happy mood))))))
```

Alle Beispiele im folgenden Abschnitt sind Teile dieses Lexikoneintrags.

Zuerst wurde das Stichwort herausgefiltert, in diesem Beispiel: "Abbruch", danach in diesem Eintrag nach Schlüsselwörtern gesucht.

Die Schlüsselwörter "tran", darin verschachtelt (in Klammern) "word" und "collocat", darin verschachtelt (in Klammern) "source" wurden ausgewertet.

Im Schlüsselwort "tran" wurden Einzelwortübersetzungen herausgefiltert, diese standen jeweils nach "word:".

Beispiel für ein 'tran'-Eintrag mit darin enthaltenem 'word':

```
tran( word: stopping)
```

Erzeugte Ausgabe in den beiden Ausgabedateien, die jeweils nur einen Eintrag in der jeweiligen Sprache enthalten:

Datei Sprache A (hier Deutsch) schon mit Tag (aus LQL direkt ausgelesen, nicht vom Tagger):

```
Abbruch <NN> Abbruch
Abbruch <NN> Abbruch
Abbruch <NN> Abbruch
Abbruch <NN> Abbruch
Abbruch <NN> Abbruch
```

Datei Sprache B (hier Englisch):

demolition  
 pulling down  
 breaking off  
 stopping  
 harm

Im Schlüsselwort "collocat" wurde das darin verschachtelte "source" und darin wiederum ein oder mehrere "targ(target(word: ...))"-Strukturen ausgewertet, die Tilde (siehe Beispiel) wurde mit dem Stichwort ersetzt. Die deutsche Übersetzung der Kollokation ist in "source" und in "targ(target(word: ...))" ist eine oder mehrere dazugehörige englische Übersetzung enthalten.

Beispiel für das Schlüsselwort 'collocat':

```
collocat( source: das hat der Fröhlichkeit keinen ~ getan targ(
  target( word: it didn't spoil the happy mood))))
```

Dies ist ein Beispiel für eine Zwischenstufe der Verarbeitung, welche auch als Datei vorliegt:

```
Abbruch*auf ~ verkaufen*to sell for demolition
Abbruch*einem Land mit ~ der diplomatischen Beziehungen drohen*
  to threaten to break off diplomatic relations with a country
Abbruch*es kam zum ~ des Kampfes*the fight had to be stopped
Abbruch*das tut unseren Interessen ~*that is detrimental to our interests
Abbruch*das hat der Fröhlichkeit keinen ~ getan*it didn't spoil the happy mood
```

Hierbei steht das Stichwort links (\* steht für tab), danach die deutsche Kollokation, gefolgt von der englischen Übersetzung. Die Tilde steht für das Stichwort und wurde später ersetzt. Diese Kollokationen und ihre Übersetzungen wurden nach fertiger Filterung als Zeilen in eigene Ausgabedateien für Sprache-A bzw. Sprache-B geschrieben.

Die erzeugte Ausgabe in diesen beiden Dateien sieht wie folgt aus:

Datei Sprache A (hier Deutsch):

```
auf Abbruch verkaufen
einem Land mit Abbruch der diplomatischen Beziehungen drohen
es kam zum Abbruch des Kampfes
das tut unseren Interessen Abbruch
das hat der Fröhlichkeit keinen Abbruch getan
```

Datei Sprache B (hier Englisch):

```
to sell for demolition
to threaten to break off diplomatic relations with a country
the fight had to be stopped
that is detrimental to our interests
it didn't spoil the happy mood
```

Ersetzungen werden durch '.or.' bzw. 'xxx/yyy' angezeigt. Sie wurden nicht verarbeitet, weil nicht immer erkennbar war, für wieviel Wörter die jeweilige Ersetzung gilt. Ersetzt wurde 'sth' durch 'something'.

Beispiel für Einträge mit '.or.' und 'xxx/yyy':

```
collocat( source: einer Sache explain: dat targ( target(
  phrase: ~ tun to harm .or. damage sth, to do (some) harm .or. damage to sth)))
collocat( source: das tut der Liebe keinen ~ targ( target(
  word: it doesn't harm their/our relationship)))
collocat( source: auf ~ stehen targ( target( phrase: to be scheduled
  .or. due for demolition, to be condemned)))
```

## 4.4 Zusammenfassung

Die DEP's liefern uns initiale Wörterbuchdaten, sowohl Einzelwort zu Einzelwortübersetzungen, Einzelwort zu Mehrwortübersetzungen als auch Mehrwort zu Mehrwortübersetzungen. Diese Mehrwortübersetzungen werden von uns als Kollokations-Prototypen in der Phrasenerkennung (s. Abschnitt 8.1) eingesetzt, um neue Kollokations-Kandidaten zu finden.

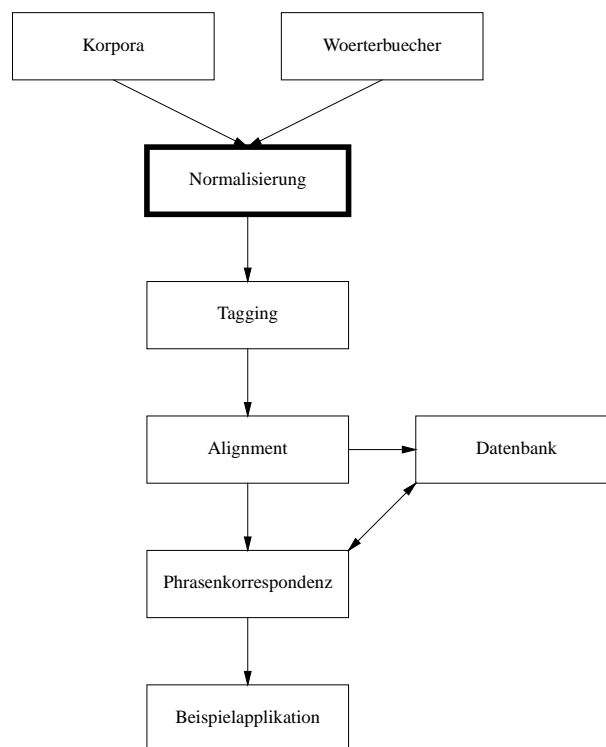
## 4.5 Ausblick

Wie bereits erwähnt, enthalten die Wörterbücher oft Zusatzinformationen. Diese können zukünftig mit großen Aufwand herausgefiltert und dem System zur Verfügung gestellt werden. Es könnten auch noch weitere Wörterbücher für unser KoKS-System verfügbar gemacht werden, dies würde aber keinen nennenswerten Vorteil mehr bringen, da die Wörterbücher nur selten neue Stichwörter enthalten, andererseits vielleicht aber neue Übersetzungen.

Deshalb entschlossen wir uns, ab Mai 2001 keine weiteren Wörterbücher in das System einzuspeisen, um unsere ganze Kraft auf die Phrasenerkennung zu konzentrieren.

# Kapitel 5

## Normalisierung



### Inhaltsverzeichnis

---

5.1	Motivation	37
5.2	Unser Vorgehen	37
5.3	Ausgabeformat	37
5.4	Filter	40
5.5	Tools	40
5.6	Ausblick	40

---

```
<DOC de-news-1996-12-23-1>
<H1>
Urteil gegen mutmasslichen Filmfaelscher Born
</H1>
Im Prozess gegen den mutmasslichen Filmfaelscher Michael Born wurde
heute das Urteil gesprochen. Born soll mit gefaelschten Filmbeitraegen vor
allem private Fernsehsender um 350.000 DM betrogen haben. Der Filmfaelscher
muss nun fuer vier Jahre hinter Gitter. Das Landgericht Koblenz verurteilte
ihn wegen Betrugs und mehrerer vorgetaeschter Straftaten. Born hatte 16
Beitraege fuer Fernsehstationen erfunden oder inszeniert.

<DOC de-news-1996-12-23-2>
<H1>
Ruehe besucht Soldaten in Bosnien
</H1>
Bundesverteidigungsminister Ruehe besuchte heute gemeinsam mit seinem
franzoesischen Amtskollegen deutsche und franzoesische SFOR-Soldaten in
Bosnien. Dabei liess sich Ruehe auch vom Sonderbeauftragten fuer den zivilen
Wiederaufbau, Bild, ueber die Lage informieren. An der SFOR-Mission
werden sich rund 3.000 Bundeswehrsoldaten beteiligen.
```

Abbildung 5.1: Ein Text aus dem de-News-Korpus im Originalzustand

## 5.1 Motivation

Korpora liegen in einer Vielzahl unterschiedlicher Formate vor, wie z.B. HTML, PDF, SGML oder ASCII. Ein Beispiel sind die in SGML vorliegenden DE-News-Texte (s. Abbildung 5.1), ein anderes die NATO-Texte (s. Abbildung 5.2). Um diese gemeinsam verarbeiten zu können, müssen sie alle in ein einheitliches Format gebracht werden. Diesen Vorgang nennt man *Normalisierung*.

## 5.2 Unser Vorgehen

Die Normalisierung geschieht vor dem Taggen, erzeugt aber schon Informationen, die für den Aligner wichtig sind. In manchen Formaten sind z.B. Absätze oder Überschriften gekennzeichnet. Mit Hilfe dieser Informationen kann man den Text partitionieren, und so Absatz für Absatz alignen. Dies führt zu einer höheren Genauigkeit und schnelleren Verarbeitung, als wenn man den ganzen Text auf einmal alignet. Deshalb sollten diese Informationen nicht verworfen werden. Für den Tagger sind sie jedoch unwichtig. Da der Tagger in einem bestimmten Modus SGML-Markup (also Elemente in spitzen Klammern <>) ignoriert, werden für den Aligner wichtige Informationen auf diese Weise vor dem Tagger versteckt.

Das Skript `preproc.py` (siehe K.2) wertet die Informationen zu jedem Korpus aus der zugehörigen XML-Datei aus (s. Kapitel Korpora, 3.6), normalisiert die Dokumente aufgrund der Angaben zum Format, ruft den Tagger und den Aligner auf und trägt das Ergebnis in die Datenbank ein.

## 5.3 Ausgabeformat

Das Ausgabeformat sollte bis auf die Alignment-Informationen keine Markup-Elemente enthalten. Das Dokumentende sowie, wenn das Eingangsformat Absätze kennzeichnet (wie z.B. in HTML), Absätze werden markiert. Ein normalisierter Beispieltext ist unter Abbildung 5.3 zu sehen.

```

<HTML>
<! (c) Copyright NATO, all rights reserved >
<! Comments to natodoc@hq.nato.int >
<HEAD>
<TITLE>NATO Kommunique - NATO HQ, Brussel - 12 April 1999</TITLE>
</HEAD>
<BODY background="../../icons/borders/press.jpg">
<A HREF="../../home.htm"><IMG SRC="../../icons/high/publicat.gif"
width=168 height=46 border=0 alt=""></A><BR>
<IMG SRC="../../icons/low/pr-e.gif" width=168 height=37 border=0
alt="">
<TABLE width=600>
<TR>
<TD width=120 valign=top>
<H5>Brüssel,<BR>12. April 1999</H5>
</TD>
<TD width=480>
<CENTER>
<hr><em>Vorläufige nichtamtliche Übersetzung</em><hr>
<H3>Die Lage in und um Kosovo</H3>
<H4>Erklärung anlässlich des außerordentlichen Treffens des
Nordatlantikrats auf Ministerebene am 12. April 1999 im
NATO-Hauptquartier in Brüssel</H4>
</CENTER>
<OL>
<LI>Die Krise im Kosovo stellt eine grundlegende Herausforderung der
Werte der Demokratie, der Menschenrechte und der Rechtsstaatlichkeit
dar, für die die NATO seit ihrer Gründung eintritt. Wir stehen geeint in
unserer Entschlossenheit, dieser Herausforderung erfolgreich zu
begegnen.
<P>
<LI>Die Bundesrepublik Jugoslawien hat wiederholt Resolutionen des
Sicherheitsrats der Vereinten Nationen verletzt. Der hemmungslose
Angriff durch jugoslawische Streitkräfte, die Polizei und
paramilitärische Kräfte unter der Leitung von Präsident Milo-sevic auf
Zivilpersonen im Kosovo hat zu einer gewaltigen humanitären Katastrophe
geführt, die auch die angrenzende Region zu destabilisieren
droht. Mehrere hunderttausend Menschen wurden rücksichtslos durch die
Behörden der Bundesrepublik Jugoslawien aus dem Kosovo vertrieben. Wir
verurteilen diese erschreckenden Verletzungen der Menschenrechte und
die blinde Gewaltanwendung durch die jugoslawische Regierung. Diese
extreme und verbrecherische, unverantwortliche Vorgehensweise, die durch
nichts zu vertreten ist, hat die Militäraktion durch die NATO notwendig
gemacht und rechtfertigt sie.
<P>
...
</OL>
<HR WIDTH=50% ALIGN=center>
<CENTER><A HREF="../../pr99e.htm">
<IMG SRC="../../icons/btn-indx.gif" border=0 ALT=" [ Go to Index ]"></A>
</CENTER>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>

```

Abbildung 5.2: Ein Text aus dem NATO-Korpus im Originalzustand

<ABSATZ>  
Brüssel,  
12. April 1999

Vorläufige nichtamtliche Übersetzung

Die Lage in und um Kosovo

Erklärung anlässlich des außerordentlichen Treffens des Nordatlantikrats auf Ministerebene am 12. April 1999 im NATO-Hauptquartier in Brüssel

Die Krise im Kosovo stellt eine grundlegende Herausforderung der Werte der Demokratie, der Menschenrechte und der Rechtsstaatlichkeit dar, für die die NATO seit ihrer Gründung eintritt. Wir stehen geeint in unserer Entschlossenheit, dieser Herausforderung erfolgreich zu begegnen.

<ABSATZ>

Die Bundesrepublik Jugoslawien hat wiederholt Resolutionen des Sicherheitsrats der Vereinten Nationen verletzt. Der hemmungslose Angriff durch jugoslawische Streitkräfte, die Polizei und paramilitärische Kräfte unter der Leitung von Präsident Milo-sevic auf Zivilpersonen im Kosovo hat zu einer gewaltigen humanitären Katastrophe geführt, die auch die angrenzende Region zu destabilisieren droht. Mehrere hunderttausend Menschen wurden rücksichtslos durch die Behörden der Bundesrepublik Jugoslawien aus dem Kosovo vertrieben. Wir verurteilen diese erschreckenden Verletzungen der Menschenrechte und die blinde Gewaltanwendung durch die jugoslawische Regierung. Diese extreme und verbrecherische, unverantwortliche Vorgehensweise, die durch nichts zu vertreten ist, hat die Militäraktion durch die NATO notwendig gemacht und rechtfertigt sie.

...

<ABSATZ>

<dokende>

Abbildung 5.3: Ein Beispiel für das Ausgabeformat

## 5.4 Filter

Es werden Filter für die Eingabeformate HTML, SGML, PDF, Postscript, Word und ASCII bereitgestellt. Letztendlich wurden jedoch nur die Filter für HTML, SGML und ASCII verwendet, der Rest verbleibt ungenutzt, weil solche Texte nicht vorkamen.

**HTML:** Es wurde die Python-Klasse `HTMLParser` erweitert, so dass für uns unwichtige Tags wie `<hr>`, `<img>` oder `<a>` ignoriert werden. Bei jedem Auftreten des `<p>`-Tags wird ein `<ABSATZ>`-Tag eingefügt.

**SGML:** Dieser Filter wird hauptsächlich für die DE-News-Texte benutzt, um deren rudimentäres Markup auszunutzen. Vor `<h1>` und nach `</h1>` werden `<ABSATZ>`-Tags eingefügt. Am Satzende werden Punkte hinzugefügt, sofern es sich nicht um eine Überschrift handelt.

**PDF:** Um die Probleme des `PDF2ASCII`-Converters zu umgehen, der mit mehrspaltigem Text und Bildern nicht umgehen kann, wird ein von Arno Erpenbeck geschriebener Umwandler von PDF nach HTML benutzt. Anschließend wird die Umwandlung mit dem `HTML-Normalisierer` weitergeführt.

**Postscript:** Wir benutzen das Skript `ps2ascii`, zu finden unter <http://www.research.digital.com/SRC/virtualpaper/pstotext.html>.

**Word:** Wir benutzen das Programm `catdoc`, zu finden unter <http://www.fe.msk.ru/~vitus/catdoc/>.

**ASCII:** Alle Texte durchlaufen nach der Konvertierung zu ASCII noch einmal ein Skript, das die Zeichen « und » durch normale Anführungszeichen (") ersetzt sowie am Dokumentende die Markierung `<DOKENDE>` anbringt. Es war angedacht, in diesem Durchgang alle Vorkommen von zwei und mehr Leerzeilen durch das `<ABSATZ>`-Tag zu ersetzen. Allerdings führt dies in der Regel zu einer unterschiedlichen Zahl von Absätzen in zwei einander zugeordneten Dokumenten. Es gelang uns nicht, dieses Problem durch Alignment der Absätze zu lösen.

## 5.5 Tools

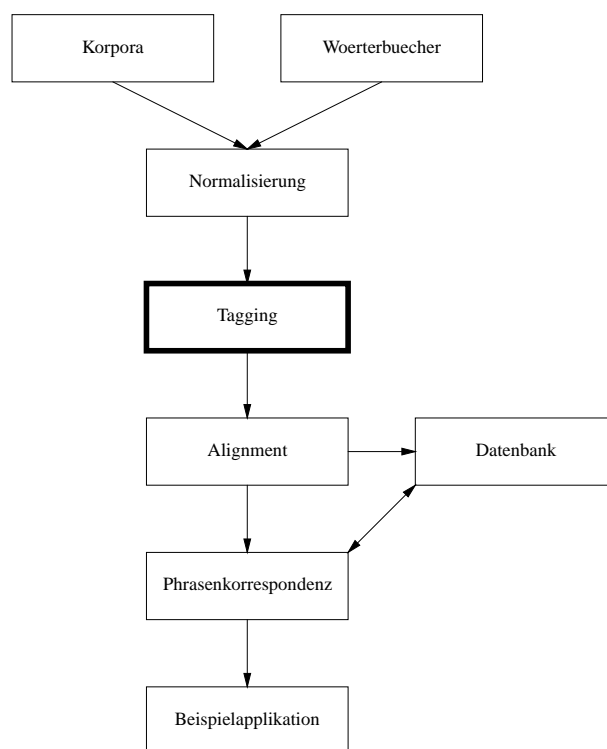
Das Tool `preproc.py` bekommt als Parameter beliebig viele XML-Dateien, die die zu verarbeitenden Korpora genauer beschreiben. Das Format dieser Dateien wird in 3.6 beschrieben, das Programm in Kapitel D.

## 5.6 Ausblick

Will man wirklich viele Korpora verarbeiten, muss man mit allgemeinerem SGML-Markup umgehen können – der `KoKS`-Konverter für SGML ist auf die DE-News-Texte spezialisiert. Gute Absatzerkennung ist wichtig, weil der Aligner besser mit kleineren Textabschnitten umgehen kann. Auch sollten alle Umwandler getestet sein. Für gutes Alignment braucht man eine gute Absatzerkennung - vielleicht lohnt es sich, eigene Umwandler für Postscript und Word zu schreiben, um dort die Absätze zu erkennen, wenn man Texte in diesen Formaten nutzen will. Auch sollte noch ein Versuch mit der Umwandlung von mehreren Leerzeilen in eine Absatzmarkierung gemacht werden.

# Kapitel 6

## Tagging



### Inhaltsverzeichnis

---

6.1	Motivation . . . . .	42
6.2	Die Tagging-Programme . . . . .	43
6.3	Ein Beispiel . . . . .	43
6.4	Bewertung und Ausblick . . . . .	46

---

## 6.1 Motivation

### 6.1.1 Grundlegendes

Tagging (zu deutsch etwa „Etikettierung“), genauer: Part-of-speech-Tagging, meint den Vorgang der Wortartbestimmung. Das kann natürlich manuell geschehen, es gibt aber diverse frei verfügbare automatische Tagger. Üblicherweise kennen sie nicht nur die „klassischen“ Wortarten, sondern reichern diese mit syntaktischem Wissen an (z.B. gibt es verschiedene Tags für Verben in unterschiedlichen Tempora), so dass die meisten Tagsets, also die Menge der von einem Tagger vergebenen Tags, um die 50 Tags umfassen.

Die üblichen Tagger arbeiten statistisch: Ein Algorithmus wird auf einem manuell getaggten Text trainiert. Dabei wird eine Parameterdatei generiert, auf deren Basis der Tagger dann unbekannte Texte taggen kann. Im Gegensatz zu Parsern arbeiten Tagger mit einer geringen Fehlerquote von unter 5% (d.h. über 95% der Wörter eines Textes werden richtig getaggt). Der Tag-Algorithmus an sich ist meist sprachunabhängig, ein sprachspezifischer Tagger entsteht erst durch das Training auf einer Datenmenge.

Oft sind Tagger zugleich auch Lemmatisierer. Sie liefern zu einem Wort und dessen Wortart auch seine Grundform mit, wie sie in einem Lexikon verzeichnet wird, soweit das möglich ist. In einigen Fällen ist die Grundform ambig; diese Ambiguität ist nur durch syntaktisches Wissen oder Weltwissen aufzulösen; ein Tagger verfügt weder über das eine noch das andere.

Für das KoKS-Projekt verwenden wir den Decision-Tree-Tagger des IMS in Stuttgart<sup>1</sup>. Er verwendet in der Version für das Deutsche das Stuttgart-Tübingen-Tagset (STTS). Es wurde vom Institut für Maschinelle Sprachverarbeitung (IMS) der Universität Stuttgart in Zusammenarbeit mit dem Seminar für Sprachwissenschaft (SfS) der Universität Tübingen entwickelt und umfasst 54 Tags (48 reine PoS-Tags und 6 zusätzliche für Interpunktion o.ä.). Die englische Version verwendet Tags des Tagsets aus dem Penn-Treebank-Projekt<sup>2</sup>. Es umfasst 48 Tags (davon 36 reine PoS-Tags). Eine Eins-zu-eins-Abbildung zwischen den Tagsets ist nicht ohne Verlust von Genauigkeit möglich. Dokumentationen der Tagsets finden sich in folgenden Publikationen:

STTS-Tagset	<a href="ftp://www.ims.uni-stuttgart.de/pub/corpora/stts_guide.ps.gz">ftp://www.ims.uni-stuttgart.de/pub/corpora/stts_guide.ps.gz</a>
Penn-Treebank-Tagset	<a href="ftp://ftp.cis.upenn.edu/pub/treebank/doc/cl93.ps.gz">ftp://ftp.cis.upenn.edu/pub/treebank/doc/cl93.ps.gz</a>
	<a href="ftp://ftp.cis.upenn.edu/pub/treebank/doc/tagguide.ps.gz">ftp://ftp.cis.upenn.edu/pub/treebank/doc/tagguide.ps.gz</a>

Ein beispielhafter Einblick in die Tagsets erfolgt in Kap. 6.3; beide Tagsets sind zudem in Anh. E beschrieben.

### 6.1.2 KoKS und Tagging

Ziel von KoKS ist es, eine Datenbank mit zweisprachigen Phrasenpaaren anzulegen, die aus parallelen Korpora extrahiert werden. Vereinfacht gesagt soll das KoKS-System ein Alignment von Phrasen und ihren Übersetzungen vornehmen. Im KoKS'schen Sinne meint „Phrase“ einen syntaktischen Komplex, die Bestandteile eines Satzes (also z.B. Nominalphrase (NP), Verbalphrase (VP) etc.). Dafür wäre idealerweise ein Parser zu verwenden, der die syntaktische Struktur eines Satzes erkennt und so Phrasen extrahieren kann. Das Problem ist, dass automatisches Parsen eine sehr komplexe Aufgabe ist: Die bisher vorhandenen Parser liefern nur in etwa 30-50% der Fälle ein eindeutiges Parseergebnis und sind somit zu unzuverlässig, um eine sinnvolle Basis für KoKS zu liefern. Darüber hinaus arbeiten sie sehr langsam.

Im KoKS-Projekt verfolgen wir daher einen anderen Ansatz: Phrasen werden auf Basis der in einem Satz aufeinander folgenden Wortarten erkannt und unter Einbeziehung der in den Lexika verzeichneten Übersetzungen einander zugeordnet. Das ist eine rudimentäre Art von Chunk-Parsing: Einzelne Wörter eines Satzes werden zu Chunks (zu deutsch etwa „Klumpen“) zusammengefasst. So entsteht für jeden Satz eine Menge von Hypothesen für Phrasen, die dann u.a. mit Hilfe von statistischen Methoden gefiltert werden soll (Genauerer hierzu s. Kap. 8).

Ein nicht unerwünschter Nebeneffekt des Tagging ist, dass die dadurch gewonnene Information zu einer zuverlässigen Erkennung von Satzenden, genauer: satzbeendender Interpunktion beiträgt. Hier hilft insbesondere die Lemmatisierung, die der Tree-Tagger ebenfalls durchführt: Ist ein auf einen Punkt folgendes Wort großgeschrieben, das zugehörige Lemma aber klein, so markiert der Punkt ein Satzende (dass dies

<sup>1</sup><http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger-de.html>

<sup>2</sup><http://www ldc.upenn.edu/doc/treebank2/treebank2.index.html>

nicht automatisch so ist, ist offensichtlich: Ordinalzahlen und Abkürzungen etwa werden ebenfalls durch einen Punkt terminiert).

## 6.2 Die Tagging-Programme

Der Prozess des Taggings gliedert sich in einzelne Teilprozesse, die von jeweils eigenen Tools übernommen werden. Sie sind zusammengefasst in den Shell-Skripten `bin/tree-tagger-german` und `bin/tree-tagger-english`. Hierbei handelt es sich um Skripte des IMS, die für KoKS angepasst und erweitert wurden. Die Teilprozesse sind im einzelnen:

- Tokenisierung
- Tagging
- Lemmatisierung
- Satzendenerkennung

In den Ablauf dieser Prozesse eingebunden ist eine Sonderzeichenrekonstruktion: Im DE-News-Korpus (vgl. Kap. 3.2.1) waren Umlaute und das  $\beta$  nicht vorhanden und auch nicht mit ASCII-Zeichen kodiert, sondern durch `ae`, `oe`, `ue` bzw. `ss` ersetzt. Dadurch ergab sich für den Tagger eine große Menge an unbekanntem Wörtern. Die Sonderzeichenrekonstruktion soll diese Zeichenpaare dort, wo es richtig ist, durch den entsprechenden Umlaut bzw.  $\beta$  ersetzen. Zwar ist dieser Vorgang nicht zwingend in den Tagprozess zu integrieren, er muss aber vor dem Taggen stattfinden. Zudem ist es sinnvoll, ihn erst nach der Tokenisierung stattfinden zu lassen, da zu diesem Zeitpunkt schon einzelne Wörter segmentiert sind.

Die Shell-Skripte `bin/tree-tagger-german` und `bin/tree-tagger-english` taggen alle als Argumente mitgegebenen Dateien. Die Ausgabe erfolgt auf `STDOUT`; werden mehrere Dateien als Argumente angegeben, wird deren Inhalt mittels `cat` aneinandergehängt und anschließend getaggt. In der Ausgabe sind dann die Dateigrenzen nicht mehr erkennbar. Sollen also mehrere Dateien getaggt werden, so bieten sich zwei Wege an: Die Dateien werden nacheinander an den Tagger übergeben, es erfolgt also ein mehrfacher Aufruf des Taggers mit jeweils einer neuen Datei. Dies lässt sich durch ein Shell-Skript leicht bewerkstelligen, und das jeweilige Taggergebnis kann durch Umlenken der Ausgabe in eine entsprechende Datei geschrieben werden. Der Nachteil hieran ist, dass der Tagger für jeden Start eine gewisse Zeit zur Initialisierung braucht (das Parameterfile wird in den Speicher geladen). Eine alternative Lösung zum Tagging mehrere Dateien ist daher, jede Datei mit einer eindeutigen Markierung zu beenden, so dass im konkatenierten Output des Taggers die Dateigrenzen erkennbar sind. Mit einem Shell- oder Perl-Skript kann dann die Ausgabe in entsprechende Teile gesplittet und in eigene Dateien geschrieben werden. Dadurch muss der Tagger nur einmal gestartet werden, die Verarbeitung erfolgt ein wenig schneller. Wird kein Argument angegeben, lesen die `tree-tagger`-Skripte von `STDIN`.

Die `tree-tagger`-Skripte „verstehen“ reinen Text als Inputformat. Eine Option, die in den Skripten standardmäßig aktiviert ist, erlaubt zudem SGML-artiges Markup: Text innerhalb spitzer Klammern wird ignoriert. Auf diese Weise kann der zu taggende Text annotiert werden, ohne dass solche Kommentare das Taggergebnis beeinflussen. Zu dieser und allen weiteren Optionen des Taggers siehe Anh. E.

## 6.3 Ein Beispiel

Die Datei `beispiel-de.asc2` soll getaggt werden. Es handelt sich um eine (fiktive) Datei aus einem parallelen Korpus. Der Inhalt der Datei sieht wie folgt aus:

---

`beispiel-de.asc2`

---

```

Mein Wochenende
<ABSATZ>
Letztes Wochenende war langweilig. Die Fete zum Ferienbeginn fiel ins
Wasser, weil die Disco abgebrannt war. Ausserdem kam auch nichts
Anstaendiges im Fernsehn.
<ABSATZ>

```

---

Es handelt sich also um eine Textdatei mit Markup, das für spätere Verarbeitungsschritte nötig ist. Das Tagging erfolgt nun mit dem Aufruf

```
$ bin/tree-tagger-german beispiel-de.asc2 >beispiel-de.tag
  reading parameters ...
  tagging ...
  done.
```

Die Datei *beispiel-de.tag* sieht wie folgt aus:

---

```

Mein      PPOSAT  mein
Wochenende  NN      Wochenende
<SATZ>
<segmentgrenze>
<ABSATZ>
Letztes NN      Letzte
Wochenende  NN      Wochenende
war        VAFIN  sein
langweilig ADJD   langweilig
.          SATZ-P .
<SATZ>
<segmentgrenze>
Die        ART    d
Fete       NN      Fete
zum        APPRART zum
Ferienbeginn NN      Ferienbeginn
fiel       VVFIN  fallen
ins        APPRART ins
Wasser     NN      Wasser
,          $,      ,
weil       KOUS   weil
die        ART    d
Disco      NN      Disco
abgebrannt VVPP   abbrennen
war        VAFIN  sein
.          SATZ-P .
<SATZ>
<segmentgrenze>
Außerdem   ADV      außerdem
kam        VVFIN  kommen
auch       ADV      auch
nichts     PIAT   nichts
Anständiges NN      Anständige
im         APPRART im
Fernsehn   NN      <unknown>
.          SATZ-P .
<SATZ>
<segmentgrenze>
<ABSATZ>

```

---

Das Ausgabeformat sieht also wie folgt aus: Jedes Token des Ausgangstextes steht auf einer eigenen Zeile. Es wird gefolgt von seinem Part-of-Speech-Tag und seiner Grundform, dem Lemma. Die Delimiter zwischen diesen Angaben sind Tabulatoren.

Der Tagger rekonstruiert Sonderzeichen („Ausserdem“ wird zu „Außerdem“); Tippfehler werden aber nicht erkannt: „Fernsehn“ wird nicht korrigiert, daher ist das Lemma unbekannt. Trotzdem wird in diesem Fall das Wort richtig getaggt.

Die Tags sind in gewissen Grenzen selbsterklärend: Ein mit N beginnendes Tag beispielsweise bezeichnet ein Nomen. Unterschieden wird hier zwischen Eigennamen (NE) und sonstigen Nomen (NN). Ein Verb-Tag beginnt mit einem V. Der folgende Buchstabe unterscheidet zwischen Vollverb (V), Hilfsverb (A) und Modalverb (M). Es folgen Angaben, ob es sich um ein finites (FIN) oder infinites (INF) Verb handelt, ob es im Imperativ steht (IMP) oder als Partizip Perfekt auftritt (PP) oder einen Infinitiv mit „zu“ bildet (ZU). Die Zeile

```
fiel VVFIN fallen
```

sagt also, dass „fallen“ hier als finites Vollverb (VVFIN) – nämlich als „fiel“ – auftritt. Unterschieden wird nicht zwischen Singular und Plural, auch verschiedene Tempusformen werden unter einem Tag zusammengefasst – eine vollständige morphologische Analyse leistet der Tagger also nicht.

Analog soll das englische Gegenstück *beispiel-en.asc2* getaggt werden. Der Inhalt sieht wie folgt aus:

---

```
beispiel-en.asc2
```

---

```
My weekend
<ABSATZ>
Last weekend was boring. The school's out party was called off. The
club had burned down. Also, there was nothing on the telly.
<ABSATZ>
```

---

Der Aufruf des Programms lautet analog:

```
$ bin/tree-tagger-english beispiel-en.asc2 >beispiel-en.tag
  reading parameters ...
  tagging ...
  done.
```

Der getaggte Output ist:

---

```
beispiel-en.tag
```

---

```
My      PP$      My
weekend NN      weekend
<SATZ>
<segmentgrenze>
<ABSATZ>
Last    JJ       last
weekend NN      weekend
was     VBD       be
boring  JJ       boring
.       SATZ-P   .
<SATZ>
<segmentgrenze>
The     DT       the
school  NN      school
's      VBZ       be
out     IN       out
party   NN      party
was     VBD       be
called  VBN      call
off     RP      off
.       SATZ-P   .
<SATZ>
<segmentgrenze>
The     DT       the
club    NN      club
had     VBD      have
```

```

burned VBN      burn
down   RP       down
.      SATZ-P   .
<SATZ>
<segmentgrenze>
Also   RB       also
'      '        '
there  EX       there
was    VBD      be
nothing NN      nothing
on     IN       on
the    DT       the
telly  NN       telly
.      SATZ-P   .
<SATZ>
<segmentgrenze>
<ABSATZ>

```

---

Das Ausgabeformat hat die gleiche Form wie das des deutschen Taggers. Auf den ersten Blick ist aber deutlich, dass sich die Tagsets unterscheiden. Ein Artikel hat hier das Tag DT (im STTS: ART), Verben werden anders kategorisiert (es wird z.B. unterschieden zwischen einem Verb in Grundform (VB) und einem Verb in past tense (VBD)). Auch hier ist die Namensgebung an den Wortarten orientiert und hierarchisch organisiert.

## 6.4 Bewertung und Ausblick

Das Taggen mit Vor- und Nacharbeiten liefert insgesamt gute Ergebnisse. Tokenisierer und Tagger arbeiten zuverlässig:

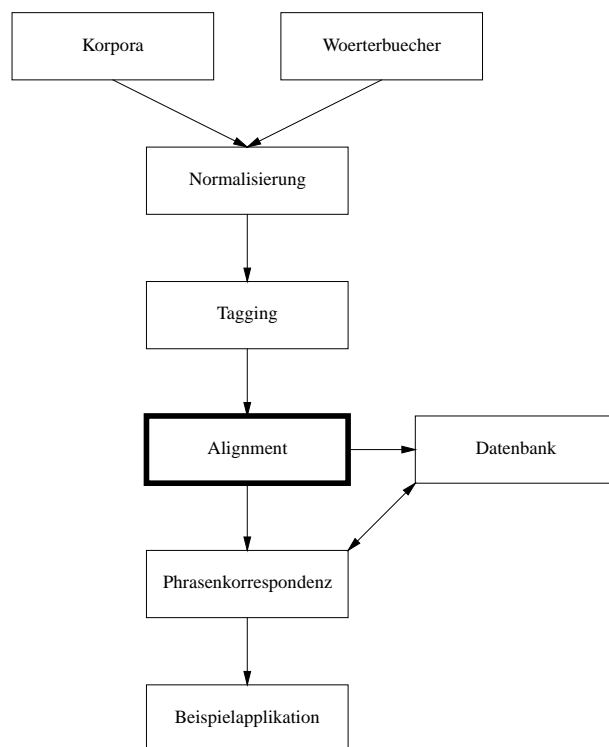
„[The Tree-Tagger] achieves 96.36 % accuracy on Penn-Treebank data which is better than that of a trigram tagger (96.06 %) on the same data.“ (Schmid, 1994, S. 1)

Auch die anderen Tools (vgl. Anh. E) arbeiten mit zufrieden stellenden Ergebnissen. Verbesserungen können im Einzelnen sicher noch erzielt werden; hier geht aber eine Kosten/Nutzen-Abwägung sicherlich nicht immer zu Gunsten des Nutzen aus.

Der Tagger kennt einige Wörter, insbesondere Eigennamen, nicht. Diese werden zwar oft korrekt getaggt, aber nicht immer; zudem fehlt für unbekannte Wörter das Lemma. Nach Abschluss der Korpusbeschaffung und -vorverarbeitung bietet es sich also an, alle unbekanntes Wörter herauszusuchen und sie – mit entsprechenden Informationen – zur Trainingsdatenbasis des Taggers hinzuzufügen, um ein verbessertes Parameterfile zu erstellen. Das kann aber nur in Zusammenarbeit mit dem IMS in Stuttgart erfolgen.

# Kapitel 7

## Alignment



### Inhaltsverzeichnis

---

7.1	Motivation	48
7.2	Alignment-Grundlagen	48
7.3	Alignment in KoKS	55
7.4	Alignment-Tools des KoKS-Projektes	70
7.5	Ausblick	74

---

## 7.1 Motivation

In KoKS sollen Phrasen und ihre Kollokativität durch die Übersetzung bestimmt werden. Dazu müssen die Übersetzungsentsprechungen im bilingualen Text (Bitext) für möglichst kleine Einheiten bekannt sein. Das parallele Korpus von KoKS enthält nach der Normalisierung und dem Taggen über 7800 Bitexte, die als Dateipaar vorliegen. I.d.R. setzen sie sich jeweils aus einem Dokument (EU) oder mehreren Artikeln (de-news) zusammen. Die Texte sind auf jeder Sprachseite getrennt in Absätze und Sätze gegliedert. Es gibt aber noch keine Informationen darüber, welche Sätze sich entsprechen. Diese Entsprechungen werden durch das Satzalignment zusammengestellt. Erst wenn die sich entsprechenden Sätze der Bitexte von KoKS identifiziert sind, kann die Phrasenerkennung beginnen, wie sie im Kapitel 8 beschrieben wird.

Ursprünglich wollten wir den im Quellcode verfügbaren Aligner aus Gale und Church (1993) verwenden. Nachdem einige offensichtliche Fehler im Quellcode behoben waren, traten aber immer wieder Probleme bei der Segmentierung der Ein- und Ausgabe auf. Z.B. stellte sich sehr spät heraus, dass der Aligner andere Sätze oder Absätze verwirft, wenn in der Eingabe ein leerer Satz oder Absatz auftritt.<sup>1</sup>

Zugleich zeigten die ersten Konzepte zur Phrasenkorrespondenz (Kapitel 8), dass ein großer Teil der Komponenten eines Aligners dort wiederverwertet werden könnte. Wir haben daher einen eigenen Aligner entwickelt. Er ist speziell an das getaggte Format in KoKS angepasst, d.h. er kann die vom Tagger annotierten Informationen für das Alignment nutzen. Des Weiteren kann der Aligner auf die bisherigen Einträge in der KoKS-Datenbank zugreifen und so z.B. die Wörterbücher zur Identifikation der Übersetzungsentsprechungen verwenden.

Im Folgenden werden die theoretischen Grundlagen des Alignments erläutert, die von uns gewählten Verfahren genau beschrieben und die im KoKS-Projekt entwickelten Alignmenttools vorgestellt. Schließlich enthält dieses Kapitel noch einen Ausblick auf weitere Verbesserungsmöglichkeiten und alternative Verfahren.

## 7.2 Alignment-Grundlagen

Eine Menge von Text zusammen mit ihrer Übersetzung in eine andere Sprache wird als bilingualer bzw. paralleler Korpus oder auch als Bitext bezeichnet. Es handelt sich um moderne Verwandte des Steins von Rosetta. Parallele Korpora stellen eine wichtige linguistische Ressource dar — sie dienen beispielsweise als Grundlage für die Erstellung bilingualer Wörterbücher oder Translation Memories. Ein Ausschnitt aus einem deutsch-englischen Bitext ist in Abbildung 7.1 zu sehen.

### 7.2.1 Übersetzungsrelationen explizieren — Alignment

Um parallele Korpora als Basis für linguistische Unterfangen nutzbar zu machen, müssen Entsprechungen von Sätzen in Sprache 1 zu ihren Übersetzungen in Sprache 2 expliziert werden. Dabei ist zu beachten, dass in der Übersetzung eines Textes kurze Sätze zu langen zusammengefasst, lange Sätze in kurze aufgeteilt, Sätze hinzugefügt oder entfernt, sowie die Reihenfolgen von Sätzen vertauscht sein können. Beim Alignment werden daher nicht einzelne Sätze einander zugeordnet, sondern Mengen von Sätzen. Die häufigsten Zuordnungen sind dabei allerdings 1:1-Zuordnungen. Abgesehen von Überkreuzungen / geänderten Satzreihenfolgen lassen sich die oben genannten Phänomene als n:m-Zuordnungen beschreiben. Ein bei der Übersetzung aufgeteilter Satz könnte beispielsweise zu einer 1:2-Zuordnung führen. Dieser Zuordnungsvorgang wird als Satz-Alignment bezeichnet, er verpackt alle Sätze beider Hälften des parallelen Korpus in solche n:m-Zuordnungen, so dass eine kontinuierliche, lückenlose Liste von Satz-Zuordnungen entsteht. Jede dieser Zuordnungen enthält also beliebig viele Sätze (möglicherweise keinen) in Sprache 1 und deren Übersetzung in Sprache 2 — das Ausgangsmaterial für linguistische Weiterverarbeitung. Abbildung 7.2 zeigt ein maschinell erzeugtes Alignment des Bitextes aus Abbildung 7.1. Die Zahlen in eckigen Klammern verzeichnen die jeweiligen Satznummern. Die Abbildungen 7.3 und 7.4 zeigen analog dazu schwierigere Fälle als 1:1-Alignments. Unser Aligner hat die deutschen Sätze 31 und 32 dem englischen Satz 30 zugeordnet. Der deutsche Satz 32 könnte aber ebenso zusammen mit dem deutschen Satz 33 dem englischen Satz 31 als zugehörig betrachtet werden. Eine mögliche Lösung wäre es, die beiden Zuordnungen zu einer größeren zusammenzufassen.

<sup>1</sup>Da zur gleichen Zeit an der Normalisierung und am Tagger gearbeitet wurde, traten leere Sätze und Absätze unter verschiedenen Bedingungen auf.

Schwere Unwetter in Norddeutschland und Bayern

Schwere Gewitter haben in der Nacht in Hamburg, Schleswig-Holstein und Niederbayern zu Ueberschwemmungen gefuehrt. Im Norden war besonders der Kreis Pinneberg betroffen. Dort stand das Wasser in einigen Strassen bis zu einem Meter hoch. Im bayerischen Straubing und Landau wurden innerhalb kurzer Zeit ganze Doerfer ueberflutet und mehrere Strassen von abgerutschten Haengen verschuettet. Die Hoehe des Schadens ist noch nicht absehbar.

FDP-Chef: Es wird keine Steuererhoeungen geben

FDP-Chef Gerhardt hat sich dafuer verbuergt, dass es keine Steuererhoeungen zur Loesung der derzeitigen Haushaltsprobleme geben wird. In einem Interview mit der BILD-Zeitung sagte Gerhardt, in diesem Punkt werde die FDP nicht wackeln oder wanken. Eine Verschiebung der Senkung des Solidarzuschlags lehnte Gerhardt ebenso ab wie Einschnitte beim Arbeitslosengeld. Bundeswirtschaftsminister Rexrodt forderte im Zusammenhang mit den Haushaltsproblemen weitere Privatisierungen von Bundesbesitz. Rexrodt sagte, der Haushaltsdruck biete die einmalige Chance, Hemmnisse zu ueberwinden, die bisher Privatisierungen behindert haetten. Der FDP-Abgeordnete Koppelin nannte die Beschaffung des Eurofighters nicht mehr finanzierbar und forderte Bundesfinanzminister Waigel auf, seine so Koppelin woertlich "suedamerikanische Buchfuehrung" zu beenden.

Bundesregierung begruesst DGB-Vorschlag

Heavy storms in northern Germany and Bavaria

Heavy thunderstorms last night have caused floodings in Hamburg, Schleswig-Holstein, and Lower Bavaria. In the North, the District of Pinneberg was most heavily affected. There, the water was up to 1 m high in some streets. In Straubing and Landau, Bavaria, complete villages were flooded within minutes and several roads covered by landslides. The cost of the damages has not been estimated yet.

FDP leader: No tax increases

FDP leader Wolfgang Gerhardt has vouched for that there will be no tax increases to solve the current budget problems. In an interview with the tabloid BILD Gerhardt said the Liberals would neither wiggle nor wobble on this point. He rejected a delay of reducing the solidarity tax as much as cuts in unemployment benefits. Regarding the current budget problems Economics Minister Guenter Rexrodt called for privatization of more government companies. Rexrodt said the budget pressure presents a unique chance to overcome hitches that have obstructed privatizations so far. FDP member of parliament Koppelin said the purchase of the Eurofighter jets can no longer be financed and called on Finance Minister Theo Waigel to stop his, as he called it, "South-American book-keeping".

Abbildung 7.1: Ausschnitt aus dem parallelen Korpus "DE-News"

L1	L2
[0]:Schwere Unwetter in Norddeutschland und Bayern	[0]:Heavy storms in northern Germany and Bavaria
[1]:Schwere Gewitter haben in der Nacht in Hamburg , Schleswig-Holstein und Niederbayern zu Überschwemmungen geführt .	[1]:Heavy thunderstorms last night have caused floodings in Hamburg , Schleswig-Holstein , and Lower Bavaria .
[2]:Im Norden war besonders der Kreis Pinneberg betroffen .	[2]:In the North , the District of Pinneberg was most heavily affected .
[3]:Dort stand das Wasser in einigen Straßen bis zu einem Meter hoch .	[3]:There , the water was up to 1 m high in some streets .
[4]:Im bayerischen Straubing und Landau wurden innerhalb kurzer Zeit ganze Dörfer überflutet und mehrere Straßen von abgerutschten Hängen verschüttet .	[4]:In Straubing and Landau , Bavaria , complete villages were flooded within minutes and several roads covered by landslides .
[5]:Die Höhe des Schadens ist noch nicht absehbar .	[5]:The cost of the damages has not been estimated yet .
[6]:FDP-Chef : Es wird keine Steuererhöhungen geben	[6]:FDP leader : No tax increases
[7]:FDP-Chef Gerhardt hat sich dafür verbürgt , daß es keine Steuererhöhungen zur Lösung der derzeitigen Haushaltsprobleme geben wird .	[7]:FDP leader Wolfgang Gerhardt has vouched for that there will be no tax increases to solve the current budget problems .
[8]:In einem Interview mit der BILD-Zeitung sagte Gerhardt , in diesem Punkt werde die FDP nicht wackeln oder wanken .	[8]:In an interview with the tabloid BILD Gerhardt said the Liberals would neither wiggle nor wobble on this point .
[9]:Eine Verschiebung der Senkung des Solidarzuschlags lehnte Gerhardt ebenso ab wie Einschnitte beim Arbeitslosengeld .	[9]:He rejected a delay of reducing the solidarity tax as much as cuts in unemployment benefits .

Abbildung 7.2: “DE-News”–Ausschnitt maschinell alignt

<p>Deutschland eingetroffen. Mubarak, der von seinem Aussenminister Mussah begleitet wird, wird morgen mit Bundeskanzler Kohl und Bundespraesident Herzog zusammentreffen. Themen der Gespraechе sind der stockende Friedensprozess im Nahen Osten sowie Handelsfragen und kulturelle Beziehungen zwischen beiden Laendern.</p>	<p>to be given higher priority in the German economy than to satisfy stockholders with top dividends.</p>
<p>Bundesregierung signalisiert den neuen Machthabern in Zaire Akzeptanz</p>	<p>Mubarak visits Germany</p> <p>Egypt's President Hosni Mubarak has arrived this afternoon in Germany for a two-day visit. Mubarak who is accompanied by his Foreign Minister Mussah will meet Chancellor Helmut Kohl and President Roman Herzog to talk about the Middle-East peace process, economics, and cultural relations between the two countries</p>
<p>Die Bundesregierung ist nach Angaben des Auswaertigen Amtes grundsatzlich bereit, mit den neuen Machthabern in Zaire zusammenzuarbeiten. Ein Sprecher des Auswaertigen Amtes in Bonn sagte, die Frage der Anerkennung der neuen Regierung stelle sich nicht. Die Bundesregierung erkenne lediglich Staaten, nicht aber Regierungen an. Der Aussenamtssprecher forderte rasche, freie und faire Wahlen in Zaire.</p>	<p>Germany indicates acceptance of Zaire's new leadership</p>
<p>Amnesty International: Menschenrechte nicht der Wirtschaft unterordnen</p>	<p>According to the German Foreign Ministry the German government is basically ready to collaborate with the new leaders of Zaire. A Foreign Ministry spokesperson said the question of recognizing the government was not an issue since Germany recognizes states but not governments. The spokesperson called for immediate, free, and fair elections in Zaire.</p>
<p>Die Gefangenenhilfsorganisation Amnesty International hat an die Bundesregierung appelliert, den Schutz der Menschenrechte nicht wirtschaftlichen Interessen unterzuordnen. Der wiedergewahlte Vorstandssprecher Kirchner sagte nach der Jahresversammlung der deutschen Amnesty-Sektion, die Politik muesse Menschenrechtsverletzungen offenansprechen. Kirchner kritisierte in diesem Zusammenhang die deutsche Haltung zu den Menschenrechtsverletzungen in China als zu nachlaessig. Ausserdem forderte Amnesty International die Innenminister auf, die Zwangsruueckfuehrung bosnischer Fluechtlinge grundsatzlich zu ueberdenken. Den</p>	<p>Amnesty International: Economy not to be set before human rights</p> <p>Prisoner advocate organization Amnesty International (ai) has appealed to the German government not to set economic interests above the protection of human rights. The re-elected executive spokesperson Mr. Kirchner said after the annual</p>

Abbildung 7.3: anderer Ausschnitt aus dem parallelen Korpus "DE-News"

[28]:Mubarak , der von seinem Außenminister Mussah begleitet wird , wird morgen mit Bundeskanzler Kohl und Bundespräsident Herzog zusammentreffen .	[28]:Mubarak who is accompanied by his Foreign Minister Mussah will meet Chancellor Helmut Kohl and President Roman Herzog to talk about the Middle-East peace process , economics , and cultural relations between the two countries
[29]:Themen der Gespräche sind der stockende Friedensprozeß im Nahen Osten sowie Handelsfragen und kulturelle Beziehungen zwischen beiden Ländern .	[29]:Germany indicates acceptance of Zaire 's new leadership
[30]:Bundesregierung signalisiert den neuen Machthabern in Zaire Akzeptanz	[30]:According to the German Foreign Ministry the German government is basically ready to collaborate with the new leaders of Zaire .
[31]:Die Bundesregierung ist nach Angaben des Auswärtigen Amtes grundsätzlich bereit , mit den neuen Machthabern in Zaire zusammenzuarbeiten .	[31]:A Foreign Ministry spokesperson said the question of recognizing the government was not an issue since Germany recognizes states but not governments .
[32]:Ein Sprecher des Auswärtigen Amtes in Bonn sagte , die Frage der Anerkennung der neuen Regierung stelle sich nicht .	[32]:The spokesperson called for immediate , free , and fair elections in Zaire .
[33]:Die Bundesregierung erkenne lediglich Staaten , nicht aber Regierungen an .	[33]:Amnesty International : Economy not to be set before human rights
[34]:Der Aussenamtssprecher forderte rasche , freie und faire Wahlen in Zaire .	
[35]:Amnesty International : Menschenrechte nicht der Wirtschaft unterordnen	

Abbildung 7.4: maschinell erstelltes Alignment des zweiten “DE-News”-Ausschnittes

<p>FDP-Chef Gerhardt hat sich dafür verbürgert, dass es keine Steuererhöhungen zur Lösung der derzeitigen Haushaltsprobleme geben wird. In einem Interview mit der BILD-Zeitung sagte Gerhardt, in diesem Punkt werde die FDP nicht wackeln oder wanken. Eine Verschiebung der Senkung des Solidarzuschlags lehnte Gerhardt ebenso ab wie Einschnitte beim Arbeitslosengeld. Bundeswirtschaftsminister Rexrodt forderte im Zusammenhang mit den Haushaltsproblemen weitere Privatisierungen von Bundesbesitz. Rexrodt sagte, der Haushaltsdruck biete die einmalige Chance, Hemmnisse zu überwinden, die bisher</p>	<p>FDP leader Wolfgang Gerhardt has vouched for that there will be no tax increases to solve the current budget problems. In an interview with the tabloid BILD Gerhardt said the Liberals would neither wobble nor wobble on this point. He rejected a delay of reducing the solidarity tax as much as cuts in unemployment benefits. Regarding the current budget problems Economics Minister Guenter Rexrodt called for privatization of more government companies. Rexrodt said the budget</p>
---	--

Abbildung 7.5: Korrespondenzen

### 7.2.2 Alignment und Korrespondenz

Das Alignment kann von dem ähnlichen Prozess des Auffindens von Wortkorrespondenzen abgegrenzt werden. Einige Wörter wie z.B. Eigennamen, Lehnwörter, Datums- und Zeitangaben oder auch Computer-Schlüsselwörter treten in beiden Hälften des parallelen Korpus identisch oder zumindest sehr ähnlich auf und stellen damit recht sichere Kandidaten für Korrespondenzen zwischen den Texten dar. Der Unterschied zum Alignment liegt in der erfassten Textmenge und der Genauigkeit der Aussage. Das Alignment umspannt alle Sätze des Textes und ordnet diese ihren (wahrscheinlichen) Übersetzungen zu, ohne innerhalb der Sätze Phrasen oder Wörter einander zuzuordnen. Die Korrespondenz ist auf wenige Wörter des Textes begrenzt, die identisch bzw. hochgradig ähnlich sind. Es werden keine Aussagen über zugehörige Sätze gemacht, eher sind Aussagen möglich wie: "Der Text um Wort x in Sprache 1 hat etwas zu tun mit dem Text um Wort y in Sprache 2". (Siehe Abbildung 7.5.)

### 7.2.3 Alignmentebenen

Ein Alignment kann auch auf höheren Ebenen als auf der Satzebene stattfinden — was bei der Übersetzung mit Sätzen passieren kann, kann ebenso Absätzen, Abschnitten oder Kapiteln widerfahren. Maschinelle Alignmentverfahren sind prinzipiell auf allen diesen Ebenen anwendbar. In der Praxis werden jedoch aufgrund der besseren Qualität der Ergebnisse in der Regel Sätze in bereits vorher (meist per Hand) alignten Absätzen verarbeitet. Phrasen oder Wörter innerhalb von Sätzen einander zuzuordnen erfordert z.B. aufgrund der unterschiedlichen Wortstellung zwischen zwei Sprachen und verschiedener Freiheiten bei der Übersetzung besser angepasste Verfahren. (Siehe 8.1.)

### 7.2.4 Maschinelle Verfahren

Lange Texte per Hand zu alignen ist für Menschen eine langweilige, zeitraubende Tätigkeit. Sind große bilinguale Korpora zu alignen, wird der Einsatz von maschinellen Verfahren wünschenswert, um die Verarbeitungsgeschwindigkeit zu erhöhen und Menschen zu entlasten. In vielen Fällen ist es einfach, eine Übereinstimmung zwischen zwei Sätzen festzustellen. Es existieren jedoch auch schwierige Fälle, in denen sogar einem Menschen die Entscheidung für eine passende Zuordnung trotz dessen Verständnisses der Sprachen sehr schwer fällt. Diese Fälle sind natürlich auch eine Fehlerquelle für maschinelle Verfahren. Die Qualität des maschinellen Alignments hängt entscheidend von der Beschaffenheit des Bitextes ab: je weniger Freiheiten bei der Übersetzung genommen wurden, desto besser sind die Aussichten auf ein gutes Alignment.

### 7.2.5 Abstand

In maschinellen Verfahren spielt der Abstand zwischen zwei Textsegmenten (z.B. Sätzen) eine entscheidende Rolle. Ein *Abstandsmaß* genanntes Modul liefert eine numerische Bewertung der Zusammengehörigkeit

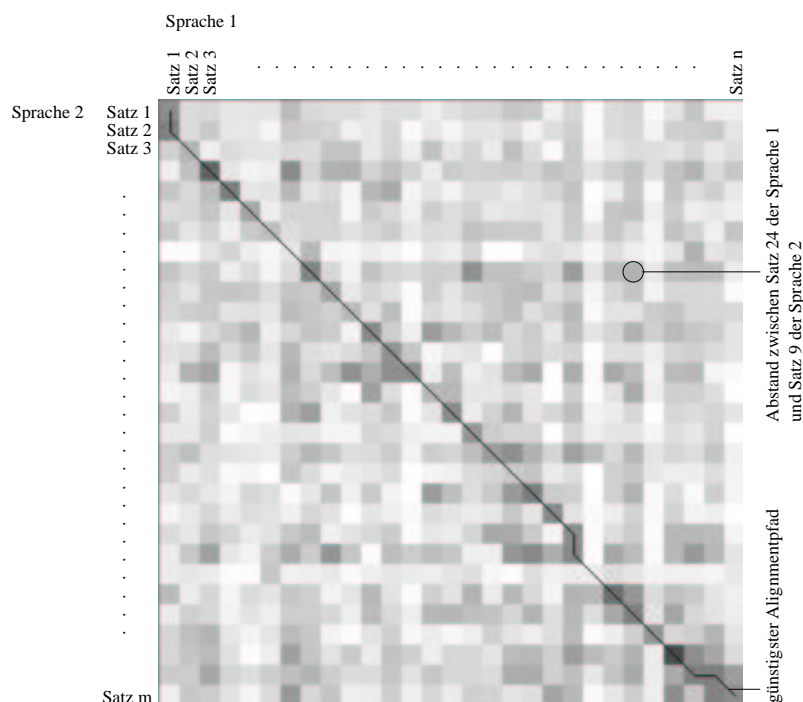


Abbildung 7.6: Abstandsmatrix und Alignment-Pfad

der beiden gegebenen Segmente (also den Abstand). Ein kleiner Abstand wird als hohe Zusammengehörigkeit interpretiert, während große Abstände als Indikator dafür angesehen werden, dass zwei Segmente sehr wenig oder sogar nichts miteinander zu tun haben.

### 7.2.6 Abstandsmatrix und Alignment-Pfad

Für alle Sätze eines Absatzes des Bitextes können ihre Abstände zueinander berechnet werden. Auf diese Weise wird eine Matrix von Abstandswerten erzeugt. Die linke obere Zelle der Matrix enthält den Abstand von Satz 1 in L1 zu Satz 1 in L2, während in der rechten unteren Zelle der Abstand von Satz n in L1 zu Satz m in L2 festgehalten wird. Jede Matrixzelle ordnet einen Satz aus L1 einem Satz aus L2 zu und gibt deren Abstand zueinander an. Ein Alignment zu finden, heißt nun, einen aus benachbarten Matrixzellen bestehenden Pfad vom ersten Satzpaar (linke obere Ecke (also erster Satz aus L1 und erster Satz aus L2)) zum letzten Satzpaar (rechte untere Ecke (also letzter Satz aus L1 und letzter Satz aus L2)) zu finden, wobei zu erwarten ist, dass dieser sich an der Diagonalen der Matrix orientieren wird. Die Anzahl der möglichen Pfade, die die diagonal gegenüberliegenden Ecken der Matrix verbinden, steigt exponentiell mit der Größe der Matrix an.

Um ein gutes (optimales) Alignment zu erhalten — also einen optimalen Pfad unter den möglichen Pfaden auszuwählen — wird an den Pfad die Forderung gestellt, minimale Kosten zu besitzen. Die Kosten des Pfades ergeben sich aus den aufsummierten Inhalten der Zellen, die den Pfad konstituieren, also aus den Abständen der Sätze, die durch den Pfad einander zugeordnet werden. Anschließend kann das durch den Pfad repräsentierte Alignment in textueller Form ausgegeben werden. Die Abbildung 7.6 zeigt eine Abstandsmatrix mit einem Alignmentpfad. Je dunkler die Zelle, desto besser die Zugehörigkeit der beiden entsprechenden Sätze.

### 7.2.7 Kontinuum von Abstandsmaß-Methoden

Es gibt verschiedene Methoden, um den Abstand zwischen zwei Segmenten zu ermitteln. Auf der einen Seite des Kontinuums befinden sich recht einfache, linguistisch arme Methoden, die die Anzahl der Buchstaben

bzw. Wörter vergleichen, wobei sie sich auf die Beobachtung stützen, dass lange Sätze im Allgemeinen in lange Sätze (und kurze in kurze) übersetzt werden. n-Gramm-basierte Methoden zählen aus, wie viele n Zeichen lange Teilstücke der Segmente übereinstimmen — gute Werte resultieren vor allem aus Eigennamen und Lehnwörtern (und anderen Wörtern, die in beiden Sprachen ähnlich oder gleich auftreten, z.B. Teile von Computerlistings). Zum anderen Ende des Spektrums hin werden die Abstandsmaße immer weiter linguistisch angereichert. Dort sind z.B. wörterbuchbasierte Übersetzungsmodelle zu finden. Solche Methoden sind sprachabhängig, während die erstgenannten unter günstigen Voraussetzungen relativ sprachunabhängig sind. Je komplizierter die linguistische Analyse, desto mehr Laufzeit muss veranschlagt werden. Auf der anderen Seite lassen linguistisch reichere Verfahren wesentlich bessere Ergebnisse erwarten.

## 7.3 Alignment in KoKS

Der im KoKS-Projekt erstellte Aligner verfolgt den oben skizzierten Ansatz, einen optimalen Pfad in einer Abstandsmatrix zu suchen. Im wesentlichen gliedert sich der Aligner in eine Komponente zur Berechnung von Abstandsmatrizen und eine zur Berechnung von optimalen Pfaden durch diese Matrizen. Ein so erzeugtes Alignment kann mit weiteren Modulen betrachtet und in die Koks-Datenbank eingetragen werden. Neben den Kommandozeilenversionen der Module gibt es eine grafische Oberfläche, die es erlaubt, Abstandsmatrizen und Alignments visuell zu untersuchen.

### 7.3.1 Abstandsmaße

Grundlage für die Berechnung der Abstandsmatrix ist immer ein Abstandsmaß. Es gibt für ein bilinguales Satzpaar einen Anhaltspunkt, ob die Sätze Übersetzungen voneinander sind. Der Abstandswert 0 bedeutet, dass nach den Kriterien des Abstandsmaßes die Sätze optimal zusammenpassen. Z.B. ist dies beim sehr einfachen Abstandsmaß TEST der Fall, sobald die Sätze die gleiche Länge haben. Ein Wert von 1 zeigt an, dass die Sätze so wenig wie nur möglich miteinander zu tun haben. In der Regel geben die Abstandsmaße Werte im Intervall  $[0, 1]$  an, die einen Vergleich verschiedener Zuordnungsalternativen erlauben.

Die Abstandsmaße betrachten nur das vorgelegte Satzpaar. Weitere Informationen, wie die Länge der Absätze, aus denen sie stammen, werden nicht berücksichtigt. Da die Reihenfolge, in der die Satzpaare einem Abstandsmaß zur Bewertung vorgelegt werden, nicht definiert ist, sollte ein Abstandsmaß auch keine Historie anlegen und auswerten. Statische Informationen, wie Statistiken, die aus Trainingsmengen gewonnen wurden, oder vorab erstellte Wörterbücher, können dagegen herangezogen werden. Zusätzlich kann ein Abstandsmaß auf die Lemma- und POS-Informationen zurückgreifen, die der Tagger bereits annotiert hat.

Im Folgenden werden die im KoKS-System vorhandenen Abstandsmaße beschrieben. Sie unterscheiden sich stark bezüglich Geschwindigkeit und Sprachabhängigkeit. Die aufwändigeren Abstandsmaße scheinen auch eine höhere Qualität zu haben. Eine gründliche Evaluation haben wir allerdings nicht durchgeführt. Da meistens 1:1-Alignments vorliegen und da bereits die einfachen Abstandsmaße eine überraschend gute Alignmentqualität ermöglichen, müssten zur Beurteilung der Qualität viele Texte ausgewertet werden.

#### Church-und-Gale-Abstandsmaß

Ursprünglich sollte in KoKS ein Aligner aus Gale und Church (1993) eingesetzt werden, der uns im Quellcode vorliegt. (Zu den Schwierigkeiten, die wir mit ihm hatten, siehe Abschnitt 12.2.1.) Er benutzt auch eine Bewertungsfunktion, die Satzpaare isoliert von ihrer Umgebung betrachtet. Da sie nur auf die Längen der Sätze und auf mathematische Funktionen zurückgreift, konnte sie leicht in KoKS reimplementiert werden.

Wesentliche Idee zu diesem Abstandsmaß ist die Wahl eines stark vereinfachten Übersetzungsmodells. Im Modell wird jedes Zeichen unabhängig von den anderen Zeichen übersetzt. Von der Anzahl der Zeichen, die die Übersetzung eines Zeichens hat, wird zur Vereinfachungen nicht verlangt, dass sie ganzzahlig und nicht negativ ist. Die Wahrscheinlichkeiten der reellwertigen Längen der Übersetzungen eines Zeichens werden durch eine Normalverteilung modelliert. Die Normalverteilung hat zwei Parameter. Die durchschnittlich zu erwartende Anzahl von übersetzten Zeichen je Zeichen der Ursprungssprache gibt der Erwartungswert  $c$  an. Der zweite Parameter  $s^2$  gibt die Varianz der Verteilung an. Durch den glockenförmigen Verlauf der Normalverteilung sind Längen um  $c$  am wahrscheinlichsten. Außerhalb des Intervalls  $[c - s, c + s]$  liegt die Länge nur mit einer Wahrscheinlichkeit von knapp 32%.

$l_1 \setminus l_2$	8	10	12	14	16	20	24	28	32	40
10	0.4048	0.2000	0.0000	0.1843	0.3466	0.5992	0.7652	0.8671	0.9267	0.9789
20	0.9071	0.8462	0.7652	0.6648	0.5476	<b>0.2798</b>	0.0000	0.2583	<b>0.4746</b>	0.7652
30	0.9878	0.9772	0.9604	0.9355	0.9006	0.7960	0.6426	<b>0.4475</b>	0.2268	<b>0.2148</b>

Tabelle 7.1: Church-und-Gale-Abstandsmaß

Für einen beliebigen Satz  $S_1$  der Sprache  $L_1$  und einen beliebigen Satz  $S_2$  der Sprache  $L_2$  soll dann aus diesen Daten und mit den Methoden der Stochastik bestimmt werden, wie wahrscheinlich es ist, dass  $S_2$  eine Übersetzung von  $S_1$  ist. Aus dem C-Quellcode ergeben sich die Formeln

$$m = \frac{l_1 + l_2/c}{2} \quad (7.1)$$

$$z = \frac{l_1 c - l_2}{s\sqrt{m}} \quad (7.2)$$

$$p = 2(1 - \Phi(|z|)) \quad (7.3)$$

wobei  $\Phi$  die Verteilungsfunktion der Standardnormalverteilung ist und  $l_1$  und  $l_2$  die Längen von  $S_1$  und  $S_2$  in Zeichen sind. (In der Reimplementation haben wir einfach die im Quellcode vorhandene Approximationsformel übernommen. Beim Vergleich mit einer auf vier Nachkommastellen genauen Tabelle traten keine Abweichungen auf.)

Die Parameter  $c$  und  $s^2$  beschreiben Mittelwert und Varianz der Normalverteilung des Übersetzungsmodells. Für Übersetzungen von Englisch nach verschiedenen anderen Sprachen werden als geeignete Werte  $c = 1$  und  $s^2 = 6,8$  genannt.

$m$  gibt den Mittelwert von der Längen  $l_1$  des Ausgangssatzes und der Länge, die man von einem Ausgangssatz erwarten würde, dessen Übersetzung die Länge  $l_2$  hat. In Gale und Church (1993) wird in (7.2) anstelle von  $m$  der Wert  $l_1$  verwendet. Dies ist seltsam, da der Quellcode im Anhang des Artikels steht.

Der Wert  $p$  gibt laut Dokumentation im Quellcode die Wahrscheinlichkeit an, dass ein englischer Satz, der aus  $l_1$  Zeichen besteht, eine Übersetzung eines Satzes ist, der aus  $l_2$  Zeichen besteht. Im Artikel ist  $p$  dagegen ein Zwischenergebnis, das als Wahrscheinlichkeit  $P(z|match)$  bezeichnet wird. Mit dem Ereignis  $z$  ist vermutlich gemeint, dass der Wert einer standardnormalverteilten Zufallsvariablen  $X$  einen Wert größer oder gleich dem Wert  $z$  annimmt. (Zumindest wird  $P(|z|)$  als  $P(|X| \geq z)$  beschrieben.) Das Abstandsmaß des Aligners aus Gale und Church (1993) wird berechnet als

$$p' = P(match)p. \quad (7.4)$$

Die Werte für  $P(match)$  wurden für 1 zu 1, 1 zu 0, 0 zu 1, 2 zu 1, 1 zu 2 und 2 zu 2 empirisch ermittelt. Motiviert wird die Formel mit der Rechnung

$$P(match|z) = P(z|match) \frac{P(match)}{c_f}, \quad (7.5)$$

wobei  $P(z|match) = p$  ist und  $c_f$  laut Artikel ein konstanter Faktor sei. (Nach dem bayesschen Satz müsste  $c_f = P(z)$  gelten.) Wir verwenden als KoKS-Abstandsmaß den Wert  $1 - p$ .

Tabelle 7.1 zeigt Abstandswerte für  $c = 1,2$  und  $s^2 = 6,8$ .  $c = 1,2$  bedeutet, dass für einen Satz mit 10 Zeichen eine Übersetzung mit 12 Zeichen erwartet wird. Der Tabellenwert 0 für  $l_1 = 10$  und  $l_2 = 12$  ist wirklich genau 0, da  $2(1 - \Phi(0)) = 2(1 - 0,5) = 1$ .  $p = 1$  kann aber nicht die Wahrscheinlichkeit dafür sein kann, dass zwei beliebige Sätze mit den Längen 10 und 12 Übersetzungen voneinander sind. Eine Wahrscheinlichkeit von 1 bedeutet, dass das Ereignis mit Sicherheit eintritt. Die genaue Bedeutung von  $p$  bleibt also offen.

**Beispiel 1** Es seien zwei Sätze à 30 bzw. 20 Zeichen aus der Sprache  $L_1$  und drei Sätze à 28, 12 bzw. 20 Zeichen aus der Sprache  $L_2$  gegeben. Die ersten beiden Sätze aus  $L_2$  haben zum ersten Satz aus  $L_1$  laut Tabelle 7.1 den Abstand 0,2148. Der Abstand der beiden 20 Zeichen langen Sätze beträgt 0,2798. Fasst

Trigramm $t$	$c(t)$
'che'	2
'ens'	2
'hen'	1
'heu'	1
'men'	1
'nsc'	2
'sch'	2

Tabelle 7.2: Trigramme des Wortes 'menschenscheu'

Alarm, Ambition, Aquarium, Au- rage, Generation, Gold, Hammer, Quiz, Quote, Rate, Rebellion, Re-  
 ra, Bank, Bastard, Blitzkrieg, Horn, Hotel, Idiot, Inferno, in- volver, Ring, Rose, Salami, Sand,  
 Block, Boxer, Bus, bitter, blind, telligent, Journal, Kiosk, Laby- Sauna, Schema, Signal, Start, Tal-  
 brutal, Chaos, Clown, Couch, rinth, Lava, Lust, Mast, Mixer, lent, Tanker, Thriller, Trauma,  
 Curry, Demonstration, Depot, De- Moment, Monster, Motto, Mu- Tumult, total, Umlaut, Uniform,  
 tailed, Dilemma, Design, Echo, Ele- se, matt, modern, Nest, Novem- Utensil, Vase, Ventilator, verbal,  
 ment, Embryo, Evolution, Export, ber, Null, normal, Opposition, Or- Villa, Vitamin, Waggon, Winter,  
 elegant, Fanfare, Film, Filter, Fin- gan, Original, Pampa, Party, Plan, Wolf, warm, wild, Yacht, Yuppie,  
 ger, Fossil, Frustration, fatal, Ga- Pose, Poster, Problem, Quadrant, Zeitgeist, Zone, Zwieback

Abbildung 7.7: Positivbeispiele mit Abstand null zur Übersetzung

man dagegen die letzten beiden Sätze von  $L_2$  zusammen und vergleicht sie mit dem zweiten Satz von  $L_1$ , dann erhält man den Abstand 0,4746. Die jeweils ersten Sätze haben den Abstand 0,4475. Man würde also die erste Zuordnung als Alignment wählen.

Da das C&G-Abstandsmaß trotz der Beschränkung auf die Längeninformatio im Aligner von Church und Gale gute Ergebnisse liefern soll, wurde in KoKS ein noch einfacheres TEST-Abstandsmaß  $\max(1, |l_1 - l_2|/10)$  implementiert, das nur den absoluten Längenunterschied auswertet. Wie der Name schon sagt, dient es aber nur Testzwecken. Ob es zum Alignen ausreicht, wurde nicht ausprobiert.

### Trigrammbasiertes Abstandsmaß

Die Trigramme einer Zeichenkette sind alle zusammenhängenden Teilketten der Länge 3. Z.B. hat die Zeichenkette 'menschenscheu' die in Tabelle 7.2 aufgeführten Trigramme. Einige treten mehrfach in der Zeichenkette auf. Die Häufigkeit des Auftretens ist in der Tabelle als  $c(t)$  für jedes Trigramm  $t$  angegeben. Insgesamt enthält die Zeichenkette  $T = 7$  verschiedene Trigramme.

Das Wort 'Klasse' besitzt die Trigramme 'Kla', 'las', 'ass' und 'sse'. Die häufige Übersetzung 'class' hat 'cla', 'las' und 'ass' als Trigramme. Sie haben die  $g = 2$  gemeinsamen Trigramme 'las' und 'ass'. Insgesamt treten in diesem Beispiel  $i = 5$  Trigramme auf. Der Quotient  $g/i$  beschreibt, welcher Anteil der Trigramme von beiden Wörtern geteilt wird. Als Trigrammabstand kann entsprechend der Wert  $1 - g/i$  benutzt werden. Wenn beide Zeichenketten identisch sind, ist der Abstand 0.

Es fällt schwer, ein Übersetzungsmodell anzugeben. Für den sinnvollen Einsatz des Trigrammabstandsmaßes ist es wichtig, dass bei der Übersetzung genügend viele Trigramme erhalten bleiben. Mit Ausnahme von Eigennamen, Zahlen, Datumsangaben usw. ist dies aber sehr selten der Fall, wie eine Stichprobe von 183354 Einträgen aus den über 400000 Wörterbucheinträgen des KoKS-Systems zeigt: 2917 Wörter, d.h. knapp 1,6%, sind identisch. Allerdings sind das hauptsächlich Fremdwörter. Abbildung 7.7 zeigt einige gewöhnliche Wörter des Deutschen, deren englische Übersetzung gleich lautet.

Unter 0,4 bleibt der Abstandswert in der Stichprobe nur bei 1659 weiteren Einträgen. In Tabelle 7.3 wurden einige Beispiele zusammengetragen. Bei der Mehrzahl resultiert der Abstand aus kleinen Unterschieden am Wortende. Interessant ist das Beispiel 'Generalmajor - major general (0,333)'. Trotz der

Deutsch $S_1$	Englisch $S_2$	$T_1$	$T_2$	$g$	$i$	Abstand
Hydratation	hydration	11	9	9	11	0,182
interplanetar	interplanetary	13	14	12	15	0,200
Fouriertransformation	Fourier transformation	21	22	19	24	0,208
Professionalismus	professionalism	17	15	14	18	0,222
quantitativ	quantitative	11	12	10	13	0,231
Individualismus	individualism	15	13	12	16	0,250
Industrialismus	industrialism	15	13	12	16	0,250
Psychophysiologie	psychophysiology	17	16	14	19	0,263
Literatur	literature	9	10	8	11	0,273
Pentagramm	pentagram	10	9	8	11	0,273
Kapitulation	capitulation	12	12	10	14	0,286
Albatros	albatross	8	9	7	10	0,300
Privileg	privilege	8	9	7	10	0,300
Kombination	combination	11	11	9	13	0,308
Individualität	individuality	14	13	11	16	0,313
Disqualifikation	disqualification	16	16	13	19	0,316
Applaus	applause	7	8	6	9	0,333
Generalmajor	major general	12	13	10	15	0,333
Frequenzmodulation	frequency modulation	18	20	15	23	0,348
Transparenz	transparency	11	12	9	14	0,357
Hormon	hormone	6	7	5	8	0,375
Profil	profile	6	7	5	8	0,375
galant	gallant	6	7	5	8	0,375
matriarchalisch	matriarchal	15	11	10	16	0,375
Textinformation	textual information	15	19	13	21	0,381
Parenthese	parenthesis	10	11	8	13	0,385

Tabelle 7.3: Positivbeispiele mit kleinem Abstand

Deutsch $S_1$	Englisch $S_2$	$T_1$	$T_2$	$g$	$i$	Abstand
Drache	dragon	6	6	2	10	0,800
Sprachtherapeut	speech therapist	15	16	5	26	0,808
Instinkthandlung	instinctive act	16	15	5	26	0,808
Weltpolitik	world politics	11	14	4	21	0,810
staatenlos	stateless	10	9	3	16	0,813
Lippenstift	lipstick	11	8	3	16	0,813
Ellbogen	elbow	8	5	2	11	0,818
Sohn	son	4	3	1	6	0,833
silbrig	silvery	7	7	2	12	0,833
Lebensqualität	quality of life	14	15	4	25	0,840
Floh	flea	4	4	1	7	0,857
mißverstehen	misunderstand	12	13	3	22	0,864
freundlich	friendly	10	8	2	16	0,875
offen	open	5	4	1	8	0,875
Nordseite	north side	9	10	2	17	0,882
sprachlos	speechless	9	10	2	17	0,882

Tabelle 7.4: Positivbeispiele mit großem Abstand

Deutsch $S_1$	Englisch $S_2$	$T_1$	$T_2$	$g$	$i$	Abstand
unbeeinflußt	uninfluenced	12	12	4	20	0,800
Ehering	wedding ring	7	10	3	16	0,813
Jagdhund	hound	8	5	2	11	0,818
Primzahl	prime number	8	12	3	17	0,824
Freiheit	freedom	8	7	2	13	0,846
Sternhaufen	star cluster	11	12	3	20	0,850
Abwesenheit	absence	11	7	2	16	0,875

Tabelle 7.5: Nicht zuordbare Beispiele mit großem Abstand

veränderten Reihenfolge der Bestandteile werden lediglich die zwei Trigramme 'alm' und 'lma' durch die drei Trigramme 'al', 'ma' und 'rg' ersetzt. Der Abstand fällt entsprechend gering aus.

Die in den Tabellen 7.3 bis 7.6 angegebenen Werte für  $g$  und  $i$  passen auf den ersten Blick nicht zu der Beschreibung in der Einleitung. Den Zeichenketten wird vorne und hinten ein Leerzeichen hinzugefügt. Dadurch treten mehr Trigramme auf. Im Beispiel 'Klasse - class' sind dies 'Kl', 'se', 'cl' und 'ss'. Die Gründe dafür werden auf den nächsten Seiten erläutert. In den Tabellen ist zusätzlich für jede Sprachseite die Anzahl der verschiedenen Trigramme angegeben ( $T_1$  bzw.  $T_2$ ). Wenn auf beiden Sprachseiten jeweils kein Trigramm doppelt vorkommt, dann sind sie zugleich die Längen der Zeichenketten ohne die zwei nachträglich hinzugefügten Leerzeichen.

Die Mehrheit der Einträge, bei denen ein Mensch noch viele Ähnlichkeiten entdeckt, erhalten einen sehr großen Trigrammabstand. In Tabelle 7.4 sind einige Beispiele aufgelistet. Hier vergrößern häufig Vokal- und Konsonantänderungen und Einfügungen (z.B. 'Weltpolitik - world politics (0,810)') den Abstand. Die Beispiele verdeutlichen die Arbeitsweise des Trigrammabstandsmaßes. Man kann z.B. bei 'staatenlos - stateless (0,813)' beobachten, dass das fehlende zweite 'a' sich nur dadurch zeigt, dass die Trigramme 'taa' und 'aat' nicht mehr enthalten sind. Beide Wörter haben die Trigramme 'sta' und 'ate'.

Tabelle 7.5 zeigt einige Beispiele, die schwer eingeordnet werden können. Einerseits sind die Wörter korrekte Übersetzungen. Andererseits ist die Ähnlichkeit deutlich geringer als bei den anderen Beispielen. Der geringe Abstand bei 'Sternhaufen - star cluster (0,850)' kommt z.B. daher, dass sie die Zeichenketten 'ster' und 'st' gemeinsam haben. Da 'ster' zwei Trigramme enthält, fällt hier also die Ähnlichkeit von 'Stern' und 'cluster' mehr ins Gewicht als die von 'Stern' und 'Star'. Ebenso sind bei 'Abwesenheit -

Deutsch $S_1$	Englisch $S_2$	$T_1$	$T_2$	$g$	$i$	Abstand
Koffer	offer	6	5	4	7	0,429
offen	offence	5	7	4	8	0,500
Wind	window	4	6	3	7	0,571
Ort	sort	3	4	2	5	0,600
Laster	later	6	5	3	8	0,625
Charme	charge	6	6	3	9	0,667
fordern	order	7	5	3	9	0,667
Hauptwort	short hauls	9	11	4	16	0,750
verbilligen	intelligent	11	11	4	18	0,778
Dickhäuter	computer	10	8	3	15	0,800
wohlgemeint	wont	11	4	2	13	0,846
Stilistik	stirring	8	8	2	15	0,867
Beherrschung	cherry	12	6	2	16	0,875

Tabelle 7.6: Negativbeispiele

absence (0,875)' nur 'sen' und ' ab' gemeinsam.

Man könnte meinen, dies sei nur ein Problem der Skalierung der Werte, d.h. dass man Werte bis zu 0,9 als geringen Abstand interpretieren müsse. Aber welche Abstandswerte ergeben sich, wenn man Zeichenketten vergleicht, die nichts miteinander zu tun haben? Modifiziert man obige Stichprobe so, dass jeweils die Übersetzung eines anderen Wörterbucheintrags gegenüber gestellt wird, dann treten immer noch Abstände zwischen 0,4 und 0,9 auf. Tabelle 7.6 zeigt einige ausgewählte Beispiele.

Im KoKS-System wird das Abstandsmaß zum Vergleich ganzer Sätze verwendet. Dies ist in zweifacher Hinsicht günstig für das Trigrammabstandsmaß. Das Beispiel 'Hauptwort - short hauls (0,750)' in Tabelle 7.6 zeigt, dass das Trigrammabstandsmaß nur lokale Eigenschaften der Zeichenketten beachtet. Die unterschiedliche Reihenfolge der gemeinsamen Trigramme 'ort' und 'hau' wird vom Abstandsmaß ignoriert. (Wie schon erwähnt, werden genaugenommen auch 'rt' und 'ha' mitgezählt.) Zwischen einem Satz und der Übersetzung bestehen häufig Unterschiede in der Wortstellung. Da die Reihenfolge der Wörter nur Trigramme beeinflusst, die mehrere Wörter überspannen (im obigen Beispiel 't h'), ist der Einfluß der Wortstellung auf den Abstand sehr gering.<sup>2</sup>

Der zweite Vorteil ist, dass Sätze i.d.R. mehrere Wörter umfassen. Neben den in den Wörterbuchstichproben beobachteten Problemwörtern sind meistens auch Eigennamen oder andere gutartige Wörter im Satz enthalten. Dadurch sind Übersetzungen wesentlich häufiger identifizierbar als bei Einzelwörtern, die in den Tabellen 7.3 bis 7.6 betrachtet wurden.

In den Beispielen wurden bereits drei Vorverarbeitungsschritte angewendet, die noch nicht genau erläutert wurden. Leicht zu erkennen ist, dass die Klein-/Großschreibung ignoriert wird. (Sonst hätten z.B. 'Design' und 'design' nicht den Abstand 0.) Dies wird erreicht, indem alle Zeichen in Kleinschreibung umgewandelt werden. Dann wird am Anfang und am Ende der Zeichenkette jeweils ein Leerzeichen eingefügt. Dadurch wird verhindert, dass Wörter, die am Rand stehen, anders behandelt werden, als die übrigen Wörter. Z.B. sind so auch die Trigramme 'de' und 'gn' enthalten, wenn das Wort 'Design' am Satzanfang oder -ende steht. Damit Wörter immer von Leerzeichen umgeben sind, müssen schließlich noch alle Satzzeichen durch Leerzeichen ersetzt werden. Trigramme, die aus dem Bereich einer Wortgrenze stammen, enthalten daher immer ein Leerzeichen.

**Beispiel 2** 'FDP-Chef Gerhardt hat sich dafür verbürgt, daß es keine Steuererhöhungen zur Lösung der derzeitigen Haushaltsprobleme geben wird.' - 'FDP leader Wolfgang Gerhardt has vouched for that there will be no taxincreases to solve the current budget problems.'

Der vorverarbeitete Text, aus dem dann die Trigramme gebildet werden, lautet ' fdp chef gerhardt hat sich dafür verbürgt daß es keine steuererhöhungen zur lösung der derzeitigen haushaltsprobleme geben

<sup>2</sup> Wenn man diesen Ansatz auf  $n$ -Gramme mit  $n > 3$  erweitert, sollte man also über die Behandlung der Wortzwischenräume nachdenken.

Trigramm $t$	$c_1(t)$	$c_2(t)$	Wörter (1)	Wörter (2)
'fd'	1	1	FDP-Chef	FDP
'ge'	2	1	Gerhardt, geben	Gerhardt
'ha'	2	1	hat, Haushaltsprobleme	has
'wi'	1	1	wird	will
'ard'	1	1	Gerhardt	Gerhardt
'at '	1	1	hat	that
'ble'	1	1	Haushaltsprobleme	problems
'che'	1	1	FDP-Chef	vouched
'der'	2	1	der, derzeitigen	leader
'dp '	1	1	FDP-Chef	FDP
'dt '	1	1	Gerhardt	Gerhardt
'er '	1	1	der	leader
'ere'	1	1	Steuererhöhungen	there
'erh'	2	1	Gerhardt, Steuererhöhungen	Gerhardt
'es '	1	1	es	taxincreases
'fdp'	1	1	FDP-Chef	FDP
'ger'	1	1	Gerhardt	Gerhardt
'har'	1	1	Gerhardt	Gerhardt
'hat'	1	1	hat	that
'lem'	1	1	Haushaltsprobleme	problems
'ng '	1	1	Lösung	Wolfgang
'obl'	1	1	Haushaltsprobleme	problems
'pro'	1	1	Haushaltsprobleme	problems
'rdt'	1	1	Gerhardt	Gerhardt
'rha'	1	1	Gerhardt	Gerhardt
'rob'	1	1	Haushaltsprobleme	problems
't h'	1	1	Gerhardt hat	Gerhardt has

Tabelle 7.7: Trigramme als Hinweis auf Wortentsprechungen

wird ' - ' fdp leader wolfgang gerhardt has vouched for that there will be no taxincreases to solve the current budget problems '.

Der deutsche Satz hat  $T_1 = 119$  verschiedene Trigramme, der englische  $T_2 = 114$ . Nur 27 Trigramme treten sowohl im deutschen als auch im englischen Satz auf. Tabelle 7.7 zeigt diese 27 gemeinsamen Trigramme des Beispielsatzes und seiner Übersetzung. Die Werte  $c_1(t)$  und  $c_2(t)$  geben die Häufigkeit des Auftretens des Trigramms  $t$  im deutschen (1) bzw. englischen (2) Satz an.

In etwa 70% der Fälle lassen sich Wörter finden, die das Trigramm enthalten und auch tatsächlich Übersetzungen voneinander sind. Von den 17 bzw. 19 Wörtern, die die Sätze umfassen, sind hierbei aber nur vier oder, wenn man 'wird - will' mitzählt, fünf Wörter beteiligt. Dies liegt an der Länge der übereinstimmenden Zeichenketten.

Der Trigrammabstand der beiden Sätze berechnet sich wie eingangs erwähnt aus einer gemeinsamen Anzahl  $g$  und einer insgesamt Anzahl  $i$  von Trigrammen nach der Formel  $1 - g/i$ . Es wird jetzt aber nicht einfach  $1 - 27/(119 + 114 - 27) = 1 - 27/206 \approx 0,869$  gerechnet, da die Häufigkeit der Trigramme noch berücksichtigt werden muss. Würde z.B. der Eigenname 'Gerhardt' in beiden Sätzen jeweils zweimal auftreten, dann würden abgesehen von den Randtrigrammen '\* g' und 't \*' keine neuen Trigramme und Gemeinsamkeiten hinzukommen. Es wäre aber wünschenswert, dass das zweite in beiden Sätzen gemeinsame Auftreten des Wortes 'Gerhardt' den Abstand verringert. Die beteiligten Trigramme sollten also doppelt zur Anzahl  $g$  der gemeinsamen Trigramme hinzugezählt werden. Damit der Quotient  $g/i$  weiterhin im Intervall  $[0, 1]$  bleibt, muss die insgesamt Zahl der Trigramme angepasst werden. Recht plausibel scheint das Maximum der Häufigkeiten zu sein, da dann im Falle der Gleichheit der Zeichenketten auch  $g = i$  gilt. Die

Trigramm $t$	$c_1(t)$	$c_2(t)$	Wörter (1)	Wörter (2)
'da'	2	0	dafür, dass	—
'de'	2	0	der, derzeitigen	—
'th'	0	3	—	that, there, the
'en'	3	0	Steuererhöhungen, derzeitigen, geben	—
'gen'	2	0	Steuererhöhungen, derzeitigen	—
'the'	0	2	—	there, the
'ung'	2	0	Steuererhöhungen, Lösung	—

Tabelle 7.8: Weitere mehrfach auftretende Trigramme

Werte

$$g = \sum_{t \text{ Trigramm}} \min(c_1(t), c_2(t)) \quad (7.6)$$

$$i = \sum_{t \text{ Trigramm}} \max(c_1(t), c_2(t)) \quad (7.7)$$

bilden also die Grundlage für den Trigrammabstand  $1 - g/i$ .

Für das Beispielsatzpaar ist  $g = 27$ , da die  $c_2(t)$  Werte für alle 27 gemeinsamen Trigramme  $t$  eins sind. Der Wert von  $i$  ist aber etwas größer als  $119 + 114 - 27$ . Für vier Trigramme  $t$  aus Tabelle 7.7 ist  $c_1(t) = 2$ , d.h.  $4(2 - 1) = 4$  kommt hinzu. Dann gibt es noch ein paar Trigramme, die die Zeichenketten zwar nicht gemeinsam haben die aber trotzdem mehrfach vorkommen. Tabelle 7.8 listet diese auf. Zu  $i$  müssen also noch  $1 + 1 + 2 + 2 + 1 + 1 + 1 = 9$  hinzugezählt werden. Damit ist  $i = 119 + 114 - 27 + 4 + 9 = 219$ . Der Trigrammabstand der Beispielsätze beträgt also  $1 - 27/219 \approx 0,877$ . Zum Vergleich: Ersetzt man auf der englischen Seite 'Gerhardt' durch 'Meyer' und 'problems' durch 'difficulties', dann erhöht sich der Abstand auf 0,944.

Leider wurde erst sehr spät ein Fehler in der Implementation (Klasse `abstand.AbstandTG_2`, siehe M.3.1) gefunden, der dazu führte, dass unter bestimmten Umständen Trigramme doppelt behandelt wurden. Viele in diesem Bericht abgedruckte Ergebnisse, insbesondere die Abstandsmatrizen, basieren daher auf andere Werte für  $g$  und  $i$ . Sie lassen sich mit

$$g = \sum_{t \text{ Trigramm}} 2 \min(c_1(t), c_2(t)) \quad (7.8)$$

$$i = \sum_{t \text{ Trigramm}} \max(c_1(t), c_2(t)) \min(2, 1 + c_1(t)c_2(t)) \quad (7.9)$$

beschreiben.

Im KoKS-System gibt es noch ein alternatives Trigrammabstandsmaß. Als Maßzahl wird die Varianz der Quotienten  $c_2(t)/c_1(t)$  für alle  $t$  mit  $c_1(t) > 0$  benutzt. Dabei wird die Symmetrie des Abstandsmaßes verletzt. Das Maß stellte sich als wenig brauchbar heraus. Es half auch nicht viel, die Varianz nicht zum Mittelwert sondern zu 1 zu berechnen.

Als letzte Anmerkung sei darauf aufmerksam gemacht, dass im KoKS-Systems Trigramme auch zur Sprachklassifikation von Dokumenten erfolgreich eingesetzt wurden. Siehe Abschnitt 3.5 Sprachidentifikation.

### Wörterbuchbasiertes Abstandsmaß

Die bisher vorgestellten Abstandsmaße werten keine lexikalischen Informationen aus. Im KoKS-System steht aber ein großes einfach strukturiertes Wörterbuch zur Verfügung. Zu jedem Wort kann leicht eine Liste von Wörtern und Mehrwortausdrücken abgefragt werden, zu der es übersetzt werden kann. Es liegt nahe, diese Informationen für ein Abstandsmaß zu verwenden.

Für jedes Wort  $w_1$  des Ausgangssatzes und jedes Wort  $w_2$  der Übersetzung wird überprüft, ob  $w_2$  in der Liste der im Wörterbuch verzeichneten Übersetzungen von  $w_1$  vorkommt. Das Nachschlagen und der

Wort	Häufigkeit
of	6285
in	3768
with	2376
for	2067
on	1584
by	1124
that	750
at	553
as	533
from	516

Tabelle 7.9: Häufigkeit einiger Wörter mit POS-Tag 'IN' in Satzpaaren, die 'mit' enthalten

Vergleich mit der Übersetzungsliste erfolgt immer über die Grundform der Wörter, die im KoKS-System bereits im Eingabetext als Lemma annotiert ist. Wird eine Übereinstimmung gefunden, dann werden beide beteiligten Wörter markiert. Probleme gibt es, wenn im Wörterbuch als Übersetzung ein Mehrwortausdruck angegeben ist. Bei Einträgen wie 'Industriellandschaft - industrial landscape' ist es sicherlich sinnvoll, zu verlangen, dass alle Wörter in der Übersetzung vorhanden sein müssen. Bei 'individualisieren - to individualize' wäre dies dagegen zu restriktiv.

**Beispiel 3** Folgende Übereinstimmung werden im Satzpaar aus Beispiel 2 gefunden: 'haben - have', 'dafür - for', 'verbürgen - vouch', 'kein - no', 'derzeitig - current' und 'werden - be'.

Wie soll nun ein Satzpaar, in dem die Übersetzungsübereinstimmungen markiert sind, bewertet werden? Es sei  $m_1$  die Anzahl der markierten Wörter im Ausgangssatz  $S_1$ ,  $m_2$  die entsprechende Anzahl in der Übersetzung  $S_2$ ,  $k_1$  die Länge von  $S_1$  und  $k_2$  die Länge von  $S_2$  in Wörtern. Es liegt nahe wie beim Trigrammabstandsmaß einen Quotienten  $g/i$  aus gemeinsamen Bestandteilen  $g$  und insgesamt vorhandenen Bestandteilen  $i$  zu bilden. Eine mögliche Herangehensweise wäre  $g = m_1 + m_2$  und  $i = k_1 + k_2$ . Alternativ könnte man in Anlehnung an das Trigrammabstandsmaß  $g = \min(m_1, m_2)$  und  $i = \max(k_1, k_2)$  setzen. Die Maßzahlen  $1 - g/i$  würden sich besonders stark unterscheiden, wenn auf einer der Sprachseiten einzelne Wörter mehrfach an Übersetzungsübereinstimmungen beteiligt sind während auf der anderen Sprachseite jeweils ein anderes Wort beteiligt ist.

Um die Symmetrie des Abstandsmaßes zu wahren, muss die gleiche Berechnung auch für die Rückrichtung von der Übersetzung zum Ausgangssatz durchgeführt werden. Ein Abstandsmaß erhält man, indem man z.B. den Mittelwert, das Minimum oder das Maximum der beiden Ähnlichkeitsmaße bestimmt. Alternativ kann man die Übersetzungsmarkierungen beider Richtungen vereinigen und dann die Abstandsrechnung durchführen.

Im KoKS-System steht aber kein reines, wörterbuchbasiertes Abstandsmaß zur Verfügung, da zumindest für die Eigennamen und Zahlen, die ja nicht im Wörterbuch stehen, eine Sonderbehandlung notwendig ist. Das im nächsten Unterabschnitt beschriebene kombinierte Abstandsmaß leistet genau das.

### Kombiniertes Abstandsmaß

Ein rein wörterbuchbasiertes Abstandsmaß kann Eigennamen, Zahlen, spezielle Fremdwörter und Fachausdrücke nicht berücksichtigen. Das Trigrammabstandsmaß hat genau bei diesen Wörtern seine Stärken. Deutliche Verbesserungen sind daher zu erwarten, wenn man den wörterbuchbasierten Abstand mit dem Trigrammabstand der Wörter kombiniert, für die über das Wörterbuch keine Entsprechung im anderen Satz gefunden wurden.

Bei den geschlossenen Wortklassen sind diese beiden Abstandsmaße aber eher ungeeignet. Der Gebrauch von Artikeln unterscheidet sich zwischen Englisch und Deutsch sehr. Präpositionen bleiben bei der Übersetzung selten erhalten. Z.B. wird 'mit' neben 'with' mit vielen anderen Wörtern übersetzt. Tabelle 7.9 zeigt die zehn häufigsten Präpositionen und subordinierende Konjunktionen, die in einer Stichprobe von

Wort	Deutsch		Wort	Englisch	
	rel.	gef. Übers.		rel.	gef. Übers.
FDP-Chef	Ja	—	FDP	Ja	—
Gerhardt	Ja	—	leader	Ja	—
hat	Ja	has	Wolfgang	Ja	—
sich	—	—	Gerhardt	Ja	—
dafür	—	—	has	Ja	hat
verbürgt	Ja	vouched	vouched	Ja	verbürgt
,	—	—	for	—	—
daß	—	—	that	—	—
es	—	—	there	—	—
keine	—	—	will	Ja	—
Steuererhöhungen	Ja	—	be	Ja	wird
zur	—	—	no	—	—
Lösung	Ja	—	taxincreases	Ja	—
der	—	—	to	—	—
derzeitigen	Ja	current	solve	Ja	—
Haushaltsprobleme	Ja	—	the	—	—
geben	Ja	—	current	Ja	derzeitigen
wird	Ja	be	budget	Ja	—
.	—	—	problems	Ja	—
.	—	—	.	—	—

Tabelle 7.10: Relevante Wörter und gefundene Übersetzungen

5000 Satzpaaren, die das Wort 'mit' enthalten, auf der englischen Seite gefunden wurden. Da 'with' nur 2376 auftritt, muss 'mit' häufig mit anderen anderen Wörtern oder gar nicht übersetzt worden sein.

Das kombinierte Abstandsmaß berechnet den Abstand zweier Sätze daher wie folgt:

1. relevante Wörter markieren
2. wörterbuchbasiertes Abstandsmaß anwenden
3. Trigrammabstandsmaß auf die restlichen relevanten Wörter anwenden
4. irrelevante Wörter mit Church-und-Gale-Abstandsmaß bewerten
5. Bewertungen kombinieren

Die relevanten Wörter sind alle Nomen, Verben, Adjektive und Adverbien, d.h. die offenen Wortklassen. Sie werden am annotierten POS-Tag erkannt. Tabelle 7.10 zeigt, welche Wörter dies im Beispiel 2 sind.

Im Wörterbuch nachgeschlagen und verglichen werden dann nur die als relevant markierten Wörter beider Sprachseiten. Neben den oben begründeten Vorteilen für die Qualität des Abstandsmaß spart dies auch Zeit. Natürlich werden dadurch weniger Übereinstimmungen gefunden. Zu Beispiel 3 sind dies 'hat - has', 'verbürgt - vouched', 'derzeitigen - current' und 'wird - be'. Der Abstand  $1 - g/i$  wird dann mit  $g = \min(m_1, m_2)$  und  $i = \max(k_1, k_2)$  wie im Unterabschnitt 'Wörterbuchbasiertes Abstandsmaß' beschrieben berechnet. Hin- und Rückrichtung werden durch die Vereinigung der Markierungen der Übersetzungsübereinstimmungen kombiniert.

Alle relevanten Wörter, für die keine Übersetzungentsprechung gefunden wird, werden je Sprache zu einer Zeichenkette zusammengesetzt, die vom Trigrammabstandsmaß bewertet wird. Im Beispiel sind dies die Zeichenketten 'FDP-Chef Gerhardt Steuererhöhungen Lösung Haushaltsprobleme geben' und 'FDP leader Wolfgang Gerhardt will taxincreases solve budget problems'.

Die übrigen, irrelevanten Wörter werden dann mit dem Church und Gale Abstandsmaß verglichen. Im Beispiel sind dies auf der deutschen Seite die zehn Wörter 'sich', 'dafür', ',', 'daß', 'es', 'keine', 'zu',

KoKS-Name	Sprachabhängigkeit	Geschwindigkeit	Anmerkung
C&G	gering	schnell	Church-und-Gale-Abstandsmaß
CG	gering	schnell	identisch mit C&G
EINS	keine	sehr schnell	Abstand immer 1
NULL	keine	sehr schnell	Abstand immer 0
TEST	keine	schnell	Brauchbarkeit nicht untersucht
TG	mittelmäßig	mittelmäßig	veraltet
TG2	mittelmäßig	mittelmäßig	Trigrammabstandsmaß
WB	sehr hoch	sehr langsam	veraltet
WB2	hoch	langsam	kombiniertes Abstandsmaß
WB2N	hoch	langsam	nur Nomen wichtig

Tabelle 7.11: Abstandsmaße im KoKS-System

'der' und '.'. Auf der englischen Seite bleiben die sieben Wörter 'for', 'that', 'ther', 'no', 'to', 'the' und '.' übrig. Eigentlich sollten sie analog zur Bestimmung des Trigrammabstandes zu einer Zeichenkette zusammengefügt werden, um dann das Church und Gale Abstandsmaß anzuwenden. Stattdessen haben wir direkt die Formeln 7.1 ff. angewendet. Dabei wurde aber versehentlich die Anzahl der Wörter eingesetzt, obwohl die Zahl der Zeichen erwartet wird.

Die drei auf diese Weise gewonnenen Abstandsbewertungen werden dann unterschiedlich gewichtet, addiert und normiert, um den Gesamtabstand zu erhalten. Als Gewichte werden jeweils die Anzahl der beteiligten Wörter zuzüglich eines Bonus eingesetzt. Einen Gewichtungsbonus in Höhe des Maximums der beiden Satzlängen in Wörtern gibt es für den wörterbuchbasierten Abstand und den Trigrammabstand, sobald die Anzahl der Wörter nicht null ist. Erhalten beide Abstände einen Bonus, dann erhält der wörterbuchbasierte Abstand einen Extrabonus in gleicher Höhe.

Seien  $i_1$  und  $i_2$  Anzahl der Wörter der beiden zu vergleichenden Sätze,  $a_1, a_2, a_3$  jeweils Church und Gale, Trigramm und wörterbuchbasierter Abstand und  $g_1, g_2, g_3$  die jeweilige Anzahl der beteiligten Wörter, dann berechnet sich der kombinierte Abstand  $a$  durch

$$b = \max(i_1, i_2) \quad (7.10)$$

$$g'_3 = g_3 + (\min(1, g_3) + \min(1, g_2 g_3))b \quad (7.11)$$

$$g'_2 = g_2 + \min(1, g_2)b \quad (7.12)$$

$$a = \frac{a_1 g_1 + a_2 g'_2 + a_3 g'_3}{g_1 + g'_2 + g'_3}. \quad (7.13)$$

Alle Wörter, die als relevant betrachtet werden, müssen im Wörterbuch nachgeschlagen werden. Da dies einige Zeit in Anspruch nimmt, gibt es ein alternatives Abstandsmaß mit dem KoKS-Namen WB2N, das nur Nomen zu den relevanten Wörtern zählt. Zu beachten ist, dass damit Verben, Adjektive und Adverbien nur noch durch das Church-und-Gale-Abstandsmaß bewertet werden. Schöner wäre hier, sie trotzdem durch das Trigrammabstandsmaß zu bewerten.

### Zusammenfassung und Ausblick

Zur Abstandsbestimmung stehen im KoKS-System verschiedene Abstandsmaße zur Verfügung, die Tabelle 7.11 auflistet. Trotz der Vielfalt gibt es noch zahlreiche Verbesserungsmöglichkeiten.

**Zeitformate** Man könnte einige Sonderbehandlungen z.B. für Zeitformate einbauen. Der Trigrammabstand der Zeichenketten '14:30 Uhr' und '02:30 pm' beträgt 0,867. Er unterscheidet sich damit nicht von zufällige auftretenden Übereinstimmungen. Wünschenswert wäre, dass die übereinstimmende Zeitpunktbenennung zu einem deutlich geringeren Abstand der Zeichenketten führt.

Abstand $a$	POS <sub>1</sub>	NEG <sub>1</sub>	Anteil	POS <sub>2</sub>	NEG <sub>2</sub>	Anteil
[0,0;0,1[	685	4	99%	685	6	99%
[0,1;0,2[	15	0	100%	15	0	100%
[0,2;0,3[	26	0	100%	26	0	100%
[0,3;0,4[	36	1	97%	36	0	100%
[0,4;0,5[	61	2	96%	61	0	100%
[0,5;0,6[	107	2	98%	107	3	97%
[0,6;0,7[	321	16	95%	321	4	98%
[0,7;0,8[	1424	81	94%	1424	25	98%
[0,8;0,9[	10437	1096	90%	10437	708	93%
[0,9;1,0[	13447	25076	34%	13447	25406	34%
{1,0}	108	389	21%	108	515	17%
[0,1]	26667	26667	50%	26667	26667	50%

Tabelle 7.12: Verteilung der Trigrammabstände

$a' = a^6$	POS <sub>1</sub>	NEG <sub>1</sub>	Anteil	POS <sub>2</sub>	NEG <sub>2</sub>	Anteil
[0,0;0,1[	1157	18	98%	1157	12	98%
[0,1;0,2[	751	40	94%	751	9	98%
[0,2;0,3[	1479	98	93%	1479	39	97%
[0,3;0,4[	2974	212	93%	2974	106	96%
[0,4;0,5[	4933	507	90%	4933	325	93%
[0,5;0,6[	5873	1766	76%	5873	1449	80%
[0,6;0,7[	4897	5569	46%	4897	5181	48%
[0,7;0,8[	2969	9340	24%	2969	9442	23%
[0,8;0,9[	1224	6949	14%	1224	7518	14%
[0,9;1,0[	302	1779	14%	302	2071	12%
{1,0}	108	389	21%	108	515	17%
[0,1]	26667	26667	50%	26667	26667	50%

Tabelle 7.13: Verteilung der Trigrammabstände zur sechsten Potenz

**Trigrammabstandsmaß** Beim Trigrammabstandsmaß häufen sich die Abstandswerte besonders im Intervall  $[0,8;1,0[$ . (Siehe Tabelle 7.12. Angegeben sind die Abstandswertverteilungen für drei Satz- bzw. Segmentpaarmengen:  $POS_1 = POS_2$  sind die Segmentpaare im EU Jahr 2000 Korpus, die unser Aligner als Alignment mit dem kombinierten Abstandsmaß gewählt hat. Man kann also davon ausgehen, dass diese Menge Segmentpaare enthält, deren Segmente Übersetzungen voneinander sind.  $NEG_1$  und  $NEG_2$  dagegen umfassen Satzpaare, die gerade nicht in der Alignmentrelation zueinander stehen. In  $NEG_1$  wurden jeweils Sätze gewählt, die möglichst nahe an alignten Segmenten stehen. In  $NEG_2$  wurden je Dokument zufällig Sätze ausgewählt und die Paare übernommen, die nicht im Alignment liegen. Je Dokument wurden jeweils gleich viele Satzpaare wie Segmentpaare ausgewählt, sodass auch die Gesamtanzahl gleich ist.) Die Abstandswerte wären gleichmäßiger im Abstandsmaßwertebereich  $[0,1]$  verteilt, wenn man sie z.B. zur sechsten Potenz erhebt, d.h. als neuen Abstandswert  $a' = a^6$  wählt, wobei  $a$  der bisherige Trigrammabstandswert ist. Tabelle 7.13 zeigt die neue Verteilung.

**Wörterbuchbasiertes und kombiniertes Abstandsmaß** Beim Übersetzen muss die syntaktische Struktur an die jeweilige Sprache angepasst werden. Dabei können auch die Wortarten wechseln. Z.B. wird in Beispiel 2 'Lösung' durch 'to solve' übersetzt. Das wörterbuchbasierte Abstandsmaß findet hier keine Entsprechung, da im Wörterbuch als Übersetzungen von 'Lösung' nur Nomen, u.a. 'denouement', 'key', 'solution' und 'termination', verzeichnet sind. Wünschenswert wäre es, entweder auf der deutschen Seite neben 'Lösung' auch 'lösen' nachzuschlagen oder auf der englischen Seite zu erkennen, dass 'solution' zu

Abstand	POS <sub>3</sub>	NEG <sub>3</sub>	Anteil
[0, 0; 0, 1[	22	0	100%
[0, 1; 0, 2[	21	0	100%
[0, 2; 0, 3[	343	0	100%
[0, 3; 0, 4[	1698	337	83%
[0, 4; 0, 5[	1211	1353	47%
[0, 5; 0, 6[	567	528	52%
[0, 6; 0, 7[	285	615	32%
[0, 7; 0, 8[	169	642	21%
[0, 8; 0, 9[	72	492	13%
[0, 9; 1, 0[	54	476	10%
{1, 0}	1	0	100%
[0, 1]	4443	4443	50%

Tabelle 7.14: Verteilung der kombinierten Abstände

Abstand	POS <sub>4</sub>	NEG <sub>4</sub>	Anteil
[0, 0; 0, 1[	8	0	100%
[0, 1; 0, 2[	16	0	100%
[0, 2; 0, 3[	352	0	100%
[0, 3; 0, 4[	1669	317	84%
[0, 4; 0, 5[	1228	1329	48%
[0, 5; 0, 6[	530	547	49%
[0, 6; 0, 7[	246	511	32%
[0, 7; 0, 8[	136	293	32%
[0, 8; 0, 9[	175	609	22%
[0, 9; 1, 0[	83	837	9%
{1, 0}	0	0	—
[0, 1]	4443	4443	50%

Tabelle 7.15: Kombinierten Abstände ohne Bonusausnahme

'solve' passt. Hier könnte man versuchen, mit einfachen Ersetzungsregeln, wie z.B. 'ung' → 'en', die Zahl der gefundenen Übereinstimmungen zu erhöhen.

Eventuell lassen sich aber auch wesentlich einfacher Verbesserungen erzielen. Die Gewichtung der einzelnen Abstandsmaße beim kombinierten Abstandsmaß muss noch überarbeitet werden. Z.B. fließen derzeit in den Abstandswert nur der Trigrammabstand und der Church-und-Gale-Abstand ein, wenn über das Wörterbuch keine Entsprechungen gefunden werden. Zwar ist der Wörterbuchabstand 1. Aber da kein Wort beteiligt ist, wird die Gewichtung auf 0 gesetzt. Eine einfache Lösung wäre dann, grundsätzlich einen Gewichtsbonus für das wörterbuchbasierte Abstandsmaß und einen kleinen Gewichtsbonus für das Trigrammabstandsmaß zu vergeben.

Tabelle 7.14 zeigt die Verteilung der Abstandswerte des kombinierten Abstandsmaßes für einen Teil der Dokumente aus dem EU Jahr 2000 (knapp 17%). Tabelle 7.15 zeigt zum Vergleich die Verteilung ohne Bonusausnahme. Zwar haben sich Abstände über 0,7 in Richtung 1,0 verschoben. Aber die Maxima der Verteilungen für alignte Sätze (POS) und nicht alignte Sätze (NEG) haben sich nicht verändert und liegen immer noch sehr dicht zusammen.

Das wörterbuchbasierte Abstandsmaß könnte mehr als um das Zweifache beschleunigt werden, indem man sich auf Einzelwortübersetzungen beschränkt. Wenn man nur Wörterbucheinträge berücksichtigt, die auf beiden Sprachseiten nur ein Wort umfassen, dann erhält man nur solche Übersetzungen, die man in der anderen Sprachrichtung auch nachschlagen würde. Man könnte sich dann auf eine Sprachrichtung beschränken, ohne dass weniger Entsprechungen gefunden werden oder die Symmetrieeigenschaft des

Abstandsmaßes verletzt würde.<sup>3</sup>

Sowohl die Qualität als auch die Geschwindigkeit des wörterbuchbasierten Abstandsmaßes können weiter verbessert werden, indem die  $n$  häufigsten Wörter nicht nachgeschlagen werden. Verben wie 'sein - be', 'haben - have' und 'müssen - must' kommen in vielen Sätzen vor und hängen mehr von der Grammatik der Sprache als vom Inhalt ab.

### 7.3.2 Abstandsmatrix

Im KoKS System wird die Abstandsmatrix nicht vollständig mit dem Abstandsmaß für alle Sätze des Eingabebites berechnet. Wir gehen davon aus, dass die Absätze in den Dokumenten bereits korrekt aligniert sind, d.h. dass bei der Übersetzung keine Absätze aufgeteilt, zusammengefügt, verworfen oder eingefügt wurden.<sup>4</sup> Daher können Sätze, die in unterschiedlichen Absätzen stehen, nicht Übersetzungen voneinander sein. Sie erhalten daher ohne aufwendige Berechnungen sofort den maximalen Abstand 1. Große Teile einer Abstandsmatrix erhalten also den Wert eins. In der Abbildung 7.8 sind dies die beiden weißen<sup>5</sup> Flächen, die durch die diagonal verlaufenden Blockstruktur getrennt werden.

Aber auch innerhalb der Absätze können Abstandsberechnungen eingespart werden. Wenn auf einer der Sprachseiten der Absatz nur einen Satz umfasst, dann kann es nur eine mögliche Zuordnung der Sätze geben.<sup>6</sup> Die entsprechenden Zellen der Matrix erhalten dann den Wert null. In der Abbildung 7.8 heben sich diese Absätze als kleine hellgrüne Blöcke ab. Neben den vielen 1:1 Blöcken kommt nur ein 1:2 Block in dieser Beispielabstandsmatrix vor.

Diese Sonderbehandlung für 1:m und 1:n Absätze bringt aber nicht nur einen Geschwindigkeitsvorteil. Der Alignmentoptimierer erhält dadurch, dass die betroffene Matrixzelle den Abstandswert 0 und alle anderen Zellen in der gleichen Zeile und Spalte den Wert 1 haben, einen besonders großen Anreiz, den Alignmentpfad durch diese Matrixzelle zu legen. Der Alignmentpfad wird also stärker an die gegebene Absatzstruktur gebunden. In der Regel sind 1:1 Absätze Überschriften oder Aufzählungselemente, die bei der Normalisierung als eigener Absatz markiert wurden, da in KoKS keine speziellen Markierungsarten vorhanden sind.

Die Berechnung der Abstandsmatrix wird weiter dadurch beschleunigt, dass auch in großen Absätzen nicht alle Abstände berechnet werden. Besonders große Absätze kommen in Korpora vor, die keine klaren Absatzmarkierungen enthalten, die bei der Normalisierung ausgewertet werden könnten. Im Extremfall besteht ein ganzes Dokument aus nur einem Absatz, der dann z.B. 1521 Sätze im Deutschen und 1393 Sätze im Englischen umfasst.  $1521 \times 1393 = 2118753$  Matrixwerte zu berechnen würde über sieben Tage dauern unter der Annahme, dass jede Matrixzelle 0,31 Sekunden braucht. Die Idee ist nun, Matrixeinträge, die mehr als  $n$  Schritte von der Verbindungslinie oben-links - unten-rechts entfernt sind, nicht mehr zu berechnen sondern den Abstand 1 einzusetzen. Für das Beispieldokument verringert sich mit  $n = 35,4$  der Aufwand um ca. 94% auf knapp zwölf Stunden. Wenn man dagegen mutig  $n = 3,54$  wählt, dann läuft das Dokument in ca. 70 Minuten durch. Mit diesem Trick gelingt es, die  $1521 \times 1393$  große Abstandsmatrix in annehmbarer Zeit zu berechnen. Allerdings steigt anschließend der Alignmentpfadoptimierer mit der Fehlermeldung „OutOfMemoryError“ aus, sodass das Alignen trotzdem nicht gelingt.

Aber auch normale Absätze, die mehr als zehn Sätze umfassen, werden durch obige Einschränkung der Berechnung deutlich schneller bearbeitet. Um auch bei kleinen Absätzen von der Beschleunigung profitieren zu können, muss  $n$  möglichst klein gewählt werden. Wählt man aber  $n$  zu klein, dann besteht die Gefahr, dass der Alignmentpfad zu sehr auf die direkte Verbindungslinie eingeengt wird und ein falsches Alignment produziert wird. Der korrekte Alignmentpfad kann um so mehr von der direkten Verbindungslinie abweichen je weiter er von dem Absatzanfang und -ende entfernt ist. Zur Mitte des Absatzes muss  $n$  also größer gewählt werden.

Im KoKS-Aligner kann dieses Verhalten durch die zwei weiteren Parameter Gradient  $g$  und Maximum  $m$  fein gesteuert werden. In Abhängigkeit von  $g$ ,  $m$  und der Entfernung der jeweiligen Matrixzelle zum Absatzanfang oder -ende, wird der Wert  $n$  angepasst.

<sup>3</sup>Oder man verzichtet auf die Symmetrie. Für das Alignment ist sie nicht essentiell.

<sup>4</sup>Da wir eine Fehlermeldung ausgeben, wenn zwei parallele Dokumente nicht die gleiche Zahl von Absätzen haben, hätten Abweichungen von dieser Annahme mit hoher Wahrscheinlichkeit auffallen müssen.

<sup>5</sup>Mavis zeigt sie schwarz an. Wir haben dies für die Abbildung geändert, um Druckerfarbe zu sparen.

<sup>6</sup>Wir gehen davon aus, dass jeder Satz eine Entsprechung in der anderen Sprache hat. 0:m oder n:0 Alignments sind also ausgeschlossen.

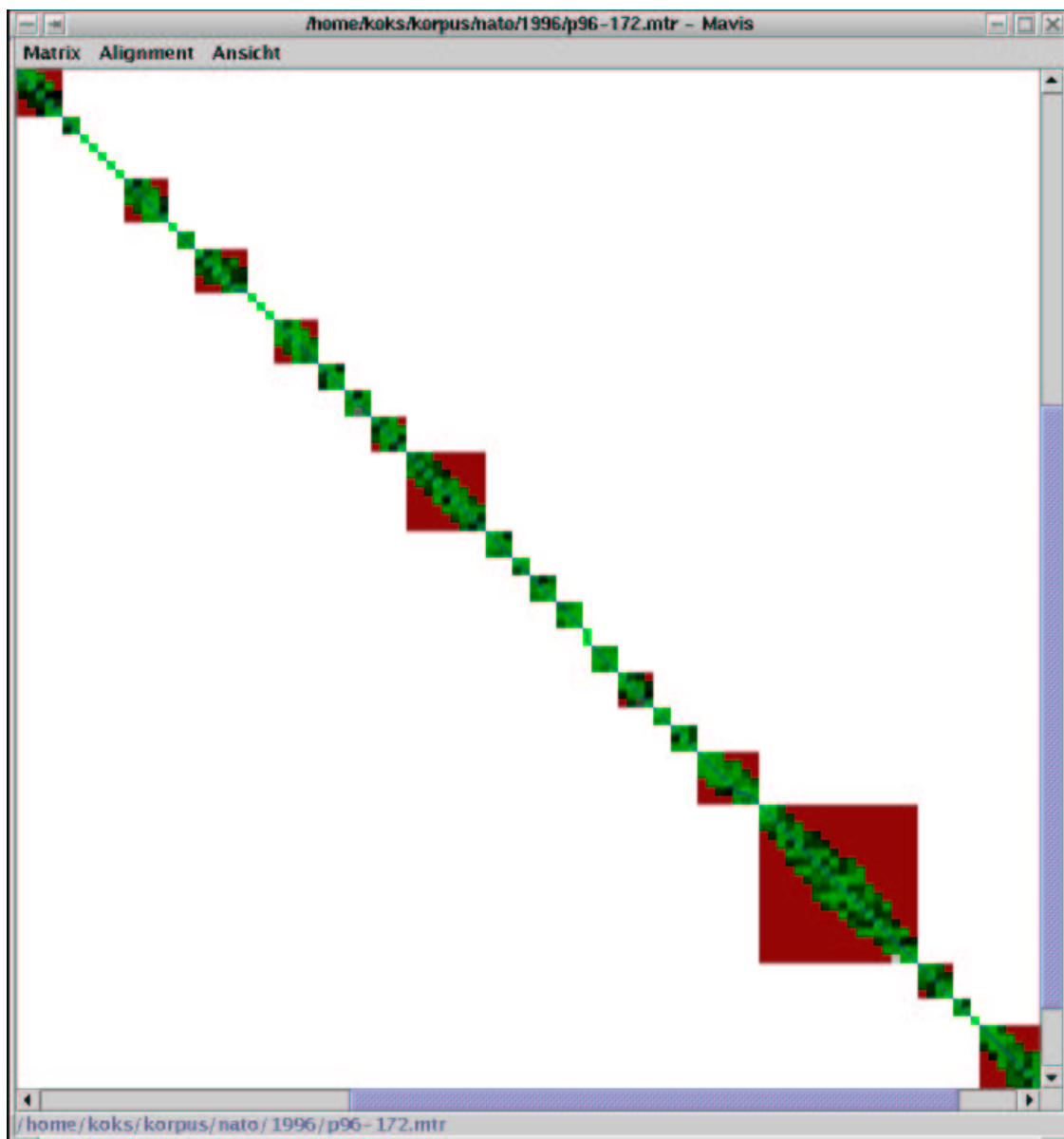


Abbildung 7.8: Grafische Darstellung einer Abstandsmatrix

Die dazu notwendigen Berechnungen vereinfachen sich, wenn man die Koordinaten der betroffenen Matrixzelle aus der Sicht der direkten Verbindungslinie von  $(0,0)$  zu  $(l_1 - 1, l_2 - 1)$  betrachtet. ( $l_1$  und  $l_2$  geben hier die Anzahl der Sätze im Absatz für die beiden Sprachen an.) Die Länge der Verbindungslinie ist dann  $l = \sqrt{(l_1 - 1)^2 + (l_2 - 1)^2}$ , und die Vektoren

$$v_1 = \frac{(l_2 - 1, l_1 - 1)}{l} \quad (7.14)$$

$$v_2 = \frac{(l_1 - 1, 1 - l_2)}{l} \quad (7.15)$$

bilden eine geeignete Orthonormalbasis, d.h.  $v_1$  und  $v_2$  stehen senkrecht aufeinander und haben die Länge eins. Die Matrixzelle an der Stellen  $w = (x, y)$  hat dann zu der Basis  $(v_1, v_2)$  die Koordinaten

$$x' = \langle w, v_1 \rangle \quad (7.16)$$

$$y' = \langle w, v_2 \rangle, \quad (7.17)$$

wobei  $\langle, \rangle$  das Standardskalarprodukt ist.

Mit anderen Worten: da  $(x', y')$  die Position der Matrixzelle aus Sicht der Verbindungslinie ist, ist der Absolutbetrag von  $y'$  die Entfernung der Matrixzelle von der Verbindungslinie. Entsprechend ist  $x'$  die Entfernung der Stelle, auf die das Lot von der Matrixzelle auf die Verbindungslinie fällt, zum Koordinatenursprung  $(0,0)$ .

Die minimale Entfernung zu den Endpunkten der Verbindungslinie ist nun einfach als

$$d = \min(x', l - x') \quad (7.18)$$

zu bestimmen. Sie wird mit dem Gradienten  $g$  multipliziert und zu  $n$  addiert. Allerdings wird das Maximum  $m$  nicht überschritten:

$$n' = \min(m, n + dg) \quad (7.19)$$

Im Fall  $|y'| > n'$  wird dann keine aufwendige Abstandsberechnung durchgeführt sondern pauschal der Maximalabstand eingesetzt.

Die Abstandsmatrix in Abbildung 7.8 wurde mit  $g = 0,12$ ,  $m = 3,53$  und  $n = 1,5$  berechnet. Die Matrixzellen mit  $|y'| > n'$  sind rotbraun hervorgehoben, damit man sie von den anderen Zellen, die ebenfalls den Maximalabstandswert erhalten, unterscheiden kann und die Blockstruktur der Absätze noch erkennen kann.<sup>7</sup>

## 7.4 Alignment-Tools des KoKS-Projektes

### 7.4.1 Kommandozeile

Das Programm `align.sh` bündelt die Schritte Abstandsmatrixberechnung und Alignmentpfadoptimierung in einen Aufruf. Das parallele Eingabedokument muss als Dateipaar im getaggen Format vorliegen. (Siehe Anhang F.3.1 Dateiformate.) Die Ausgabe erfolgt in ein neues Dateipaar, deren `<segmentgrenze>` Markierungen das Alignment angeben.

#### Aufruf

`align.sh` befindet sich im CVS-Baum im Verzeichnis `align` und wird mit folgendem Befehl gestartet:

```
$ align.sh <Dokumentpräfix> <Abstandsmaß>
```

Die Dateisuffixe für das Dokumentpaar lauten `.de.tag` und `.en.tag`. Die Ausgabe erfolgt in Dateien mit den Suffixen `.de.tag.al` und `.en.tag.al` im aktuellen Arbeitsverzeichnis.

Sind die Eingabedateien `dok1.de.tag` und `dok1.en.tag` und soll das kombinierte Absandsmaß mit dem KoKS-Namen `WB2` (siehe Tabelle 7.11) verwendet werden, lautet der Aufruf also

<sup>7</sup>Um die unterschiedliche Farbgebung zu realisieren ohne das Dateiformat ändern zu müssen wurde als Abstand 1,01 eingetragen. Der Matrixaligner und Mavis tollerieren leichte Abweichungen vom  $[0;1]$  Intervall, sodass dies unproblematisch sein sollte.

Programm	speichert zusätzlich	Anmerkung
align.sh	Abstandsmatrix	—
align_lzu1.sh	—	kopiert nur die Dokumente
align_ai.sh	Abstandsmatrix und -info	—
align_debug.sh	Abstandsmatrix, -info und Alignment	—
align_mtr.sh	—	liest vorhandene Abstandsmatrix
align_pur.sh	—	—

Tabelle 7.16: Varianten des Programms align.sh

```
$ align.sh dok1 WB2
```

Zur Kompatibilität mit dem anfangs benutzten Aligner gibt es einen alternativen Kommandozeilenauf-  
ruf, der beliebige Namen für die Eingabedateien erlaubt und die Ausgabe in das selbe Verzeichnis schreibt,  
in dem auch die Eingabedateien stehen. Der Aufruf lautet zu obigem Beispiel

```
$ align.sh -e -D "<ABSATZ>" -d "<SATZ>" dok1.de.tag dok1.en.tag
```

Zusätzlich wird die Abstandsmatrix als Datei dok1.mtr gespeichert. (Die Verwendung der normalen Suffixe  
wird automatisch erkannt. Würde z.B. der zweite Dateiname abweichend dok-en1.tag lauten, dann wäre die  
Abstandsmatrix als Datei dok1.de.tag.mtr gespeichert worden.)

### Darstellung des Alignments

Das Modul eingabe.py kann benutzt werden, um den alignten Text segmentweise auszugeben. Da es nicht  
direkt ausführbar ist, muss es über den Python-Interpreter gestartet werden:

```
$ python eingabe.py -r segment -n -i dok1.de.tag.al -j dok1.en.tag.al
```

Anmerkung: Das hier nicht beschriebene Programm alvis.py benötigt zur Alignment-Visualisierung  
ein spezielles Zwischenergebnis, das align.sh nicht speichert. Siehe Anhang.

### Varianten

Für den langen Aufruf mit sieben Parametern gibt es einige Varianten, die zusätzliche Informationen ausge-  
ben, die in Mavis angezeigt werden können. Tabelle 7.16 zeigt eine Übersicht. align\_ai.sh ist für Mavis  
sinnvoll, da dann für jede Matrixzelle angezeigt werden kann, wie der Abstandswert zu Stande kommt.  
align.sh speichert die Abstandsmatrix, damit man die Dokumente zu einem späteren Zeitpunkt, z.B. nach  
Verbesserungen am Tagger, ohne Neuberechnung der Abstandsmatrix mit Hilfe von align\_mtr.sh alignen  
kann.

### Aufbereitung für Mavis

Mavis (siehe unten) erwartet, dass die Dateinamen der Dateien, die die Informationen für Satzinfo und  
Abstandinfo enthalten, mit den Suffixen .de.tag, .en.tag und .abstandinfo aus einem Stamm gebildet werden  
können. Das Programm mtr2mavis.py überprüft dies und legt ggf. Verknüpfungen (Hardlinks) an, sodass  
die Dateien unter neuen Namen ansprechbar sind. Des Weiteren wird eine neue Abstandsmatrix gespeichert,  
die die neuen Namen im Kopfteil verwendet und ansonsten die Werte von der alten Matrix übernimmt.

Der Aufruf

```
$ mtr2mavis.py dok1-de.tag.mtr
```

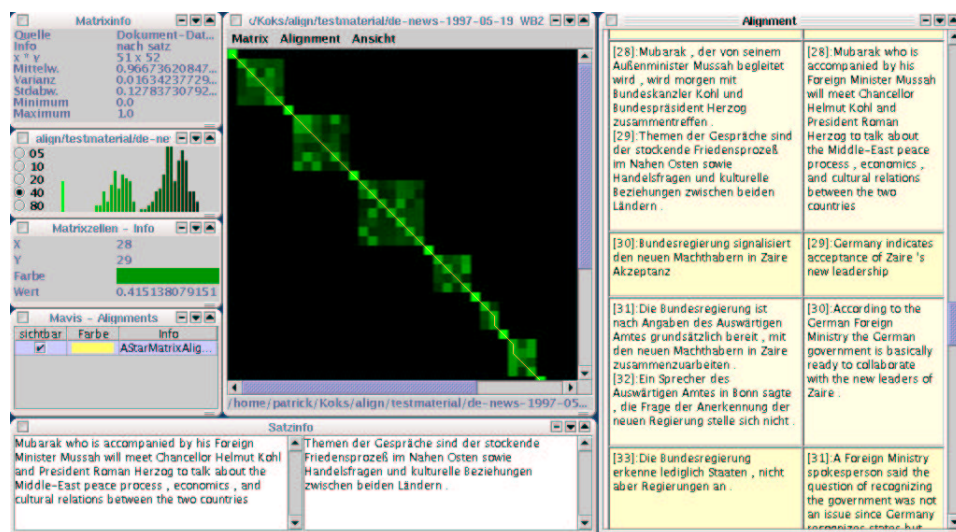


Abbildung 7.9: Visualisierungstool “Mavis”

legt Verknüpfungen `dok1-~1.de.tag`, `dok1-~1.en.tag` und ggf. `dok1-~1.abstandinfo` an und erstellt eine neue Abstandsmatrix `dok1-~1.mtr`. Wenn `mtr2mavis.py` nicht aus dem gleichen Verzeichnis heraus gestartet wird wie die Berechnung der Abstandsmatrix, dann ist das Ergebnis undefiniert. Das Programm kann auch mehrere `.mtr` Dateien in einem Aufruf bearbeiten, so dass ein ganzer Verzeichnisbaum mit `find -name "*.mtr" | xargs mtr2mavis.py` abgehandelt werden kann.

#### 7.4.2 Visualisierungstool “Mavis”

Das GUI-Tool “Mavis” (MatrixAlignmentVisualisierung) dient zur Visualisierung von Abstandsmatrizen und erlaubt die Berechnung von Alignments nach verschiedenen (matrix- und pfadorientierten) Verfahren. Abbildung 7.9 zeigt einen Screenshot des Tools.

Mavis liest eine Abstandsmatrix aus einer Datei ein und zeigt diese farbcodiert an. Mit Hilfe der Quellinformation aus der Matrixdatei (siehe Dateiformat im Anhang F.3.2) kann Mavis, wenn es aus dem gleichen Verzeichnis heraus wie der Aligner gestartet wurde, die zugehörigen `.tag` Dateien finden und so neben den Abstandswerten auch die den Abstandswert erzeugenden Satzpaare anzeigen, bzw. gefundene Alignment-Pfade in tatsächlich alignten Text in Tabellenform umsetzen. Eine Erweiterung von Joachim zeigt zusätzliche Informationen über das Abstandsmaß an. Auf diese Weise ist Mavis ein Werkzeug, das während Entwicklung und Debugging von Abstandsmaßen und Alignmentverfahren eine große Hilfe sein kann. Es ist relativ leicht, ein neues Alignmentverfahren zu implementieren und innerhalb von Mavis auszuführen. Ist man als Entwickler mit dem Alignmentverfahren zufrieden, kann es auf unkomplizierte Weise in ein Kommandozeilenprogramm verpflanzt werden.

#### Aufruf

Mavis liegt innerhalb des `align` Verzeichnisses des CVS-Baums. Hat man das Verzeichnis ausgecheckt und befindet man sich in ihm, führt

```
make dds
```

zur Kompilierung. Heraus kommt unter anderem die jar-Datei `mavis.jar`. Sie enthält alles Nötige, um Mavis auszuführen und kann daher bedenkenlos verschoben werden. Mavis wird mit dem folgenden Befehl gestartet:

```
java -jar mavis.jar
```

## Bedienung

Der erste Schritt in der Arbeit mit Mavis besteht darin, eine Abstandsmatrix aus einer Datei einzulesen. Dies geschieht über den Öffnen-Eintrag im Matrix-Menü. Dieses Menü bietet des Weiteren Möglichkeiten, die Abstandswerte in das Intervall  $[0,1 \dots 1]$  zu verschieben (Normalisieren), für eine Gleichverteilung der Werte zu sorgen (Rangfolgenclustering), die Matrix zu verkleinern (das kommt manchem Suchalgorithmus entgegen), sowie die Matrix erneut aus der Datei einzulesen und damit alle Veränderungen zurückzusetzen. Außerdem kann ein einfaches Histogramm der Matrixwerte angezeigt werden.

Der zweite Schritt besteht darin, innerhalb der Matrix einen Alignment-Pfad zu bestimmen. Für diese Aufgabe stehen derzeit folgende Algorithmen zur Verfügung:

- Best-First-Suche
- Levenshtein Minimum Edit Distance
- A\*-Suche
- Best Cell 1:1

Der Levenshtein-Algorithmus besitzt die beste Laufzeit (quadratisch), liefert jedoch schlechtere Qualität als der annehmbar performante A\*-Algorithmus. Die Best-First-Suche findet die selben Wege wie der A\*, ist jedoch wesentlich langsamer und nur für kleine Matrizen zu verwenden. In der Praxis wird daher im KoKS-Projekt der A\*-Algorithmus eingesetzt, der auch im Kommandozeilentool `al.jar` zum Einsatz kommt. Einer der Korpora (DE-News) enthielt stellenweise den Fehler, dass nicht zusammengehörige Texte die gleiche Identifikationsnummer enthielten (siehe 3.4.2). Ein "Best Cell 1:1" getauftes Verfahren sucht in einer Matrix mit Abständen auf Absatzebene nach der besten bestehenden Abstandsrelation, ordnet die entsprechenden Absätze einander zu, entfernt die Absätze aus der Matrix und verfährt mit der verkleinerten Matrix auf die gleiche Weise weiter, bis keine Zuordnungen mehr gemacht werden können. (Bei nicht-quadratischen Matrizen verbleiben zwangsläufig einige Absätze einer Sprache ohne Zuordnung, da ja nur 1:1-Zuordnungen von Absätzen erlaubt waren.) Bei diesem Vorgehen können also Überkreuzungen entstehen. Je mehr Überkreuzungen, desto höher die Wahrscheinlichkeit, dass beide Texte nichts miteinander zu tun haben.

In Mavis können mehrere Alignments auf einer Matrix parallel ablaufen. Jeder ins Leben gerufenen Alignmentvorgang präsentiert ein kleines Fenster auf dem Bildschirm, das Statusinformationen ausgibt und einen Abbruch-Knopf zur Verfügung stellt. Zum Experimentieren zeigt der A\*-Aligner ein Konfigurationspanel an, bevor er startet. Wird während laufender Alignmentvorgänge die Matrix verändert, werden die Alignments automatisch abgebrochen. Wurde ein Verfahren erfolgreich beendet, wird der gefundene Alignment-Pfad in die Liste der Alignment-Pfade (Fenster "Mavis - Alignments") eingetragen und auf der Matrix eingezeichnet.

Das Fenster "Mavis - Alignments" dient zur Verwaltung der Alignment-Pfade, sie können selektiert, ein- und ausgeblendet, sowie umgefärbt werden. Das Menü "Alignment" erlaubt es, den im Alignment-Fenster ausgewählten Pfad zu speichern, zu löschen oder textuell anzuzeigen. Es ist außerdem möglich, alle Pfade zu löschen, und gespeicherte Pfade zu laden. Letztere werden nicht wie gewohnt als Pfade sondern als Zuordnungen dargestellt. Die anfängliche Repräsentation als Pfad ist entwicklungshistorisch bedingt. Zuordnungen sind eine bessere Metapher für das Alignment.

Schließlich gibt es noch ein Menü mit der Bezeichnung "Ansicht". Hier kann die Größe eines Matrixblockes auf dem Bildschirm eingestellt werden. Darüber hinaus kann eine Linie von der linken oberen Zelle zur rechten unteren Zelle angezeigt werden. Diese Pseudo-Diagonale macht starke Abweichungen des Pfades von der Diagonalen deutlich.

Weitere Fenster der Oberfläche zeigen Informationen zur Matrix, bzw. zur Matrixzelle unter dem Mauscursor an. "Matrixzellen - Info" liefert X- und Y-Koordinate, Farbe und Abstandswert einer Zelle. "Matrixinfo" zeigt Größe, Quelle und Info sowie einige statistische Werte zur geladenen Matrix an.

Das Fenster "Satzinfo" zeigt, sofern Mavis die Quelldateien öffnen konnte, zu der Matrixzelle unter dem Mauscursor die zugehörigen Sätze an. Ist eine Datei mit Informationen zum Abstandsmaß vorhanden, wird die zur aktuellen Zelle gehörige Abstandsmaßinformation in dem Fenster "Abstandinfo" angezeigt.

Beenden lässt sich Mavis mittels des Schließen-Knopfs des Hauptfensters.

Kommandozeilenoptionen zu den einzelnen Programmen finden sich im Anhang F.

Seg.Nr.	$L_1$ : Satz	$L_2$ : Satz	Abstand	Seg.Nr.	$L_1$ : Satz	$L_2$ : Satz	Abstand
30	[ $D_{31}$ ]	[ $E_{30}$ ]	0,373	30	[ $D_{31}, D_{32}$ ]	[ $E_{30}$ ]	0,546
31	[ $D_{32}, D_{33}$ ]	[ $E_{31}$ ]	0,491	31	[ $D_{33}$ ]	[ $E_{31}$ ]	0,594
32	[ $D_{34}$ ]	[ $E_{32}$ ]	0,321	32	[ $D_{34}$ ]	[ $E_{32}$ ]	0,321

manuelles Alignment Alignment des KoKS-Aligners

Tabelle 7.17: Der Alignmentfehler aus Abbildung 7.4

$L_1 \setminus L_2$	Satz $E_{29}$	Satz $E_{30}$	Satz $E_{31}$	Satz $E_{32}$	Satz $E_{33}$
Satz $D_{30}$	0,000	1,000	1,000	1,000	1,000
Satz $D_{31}$	1,000	<b>0,373</b>	0,745	1,010	1,000
Satz $D_{32}$	1,000	<b>0,388</b>	0,504	0,414	1,000
Satz $D_{33}$	1,000	0,650	<b>0,594</b>	0,700	1,000
Satz $D_{34}$	1,000	1,010	0,853	<b>0,321</b>	1,000
Satz $D_{35}$	1,000	1,000	1,000	1,000	0,000

Abbildung 7.10: Ausschnitt aus der Abstandsmatrix zum besprochenen Dokument

## 7.5 Ausblick

### 7.5.1 Zusammenfassung

Der KoKS-Aligner kann mit  $n$  zu  $m$  Alignments auch für  $n, m > 2$  umgehen. Der modulare Aufbau des Aligners erlaubt es, verschiedene Abstandsmaße und Pfadsuchstrategien zu kombinieren. Das Visualisierungstool *Mavis* unterstützt dabei den Entwickler. Ebenso kann *Mavis* benutzt werden, um das Alignment eines Bitextes manuell zu überprüfen und dabei über das Satzinfo-Fenster Zuordnungsalternativen machen. Wünschenswert wäre hier nur noch, dass der zweitbeste Pfad einer Suche angezeigt werden kann, vorhandene Pfade manipuliert werden können und dass die Gesamtbewertung eines Pfads in einer Rangtabelle angezeigt wird.

### 7.5.2 Kritik am Konzept

Trotz des Erfolgs ist Kritik an der ausschließlichen Verwendung der Werte der Satzabstandsmatrix und an der Gleichsetzung eines Alignments mit einem Pfad durch diese Matrix berechtigt. Ein  $n$  zu  $m$  Alignment wird nicht danach beurteilt, welchen Abstand die  $n$  Sätze der einen Sprache als Ganzes zu den  $m$  Sätzen der anderen Sprache haben, sondern danach, welche Summe die  $n + m - 1$  Einzelabstände haben, die auf dem Weg des Alignmentpfades liegen. Der einzige Fall eines falschen 1:2 Alignments, den wir genauer untersucht haben, belegt, dass dabei tatsächlich gegensätzliche Bewertungen entstehen können. Es handelt sich dabei um das de-news Dokument 1997-05-19. Der relevante Ausschnitt kann in Abbildung 7.4 auf Seite 52 nachgelesen werden. Abbildung 7.10 zeigt einen Ausschnitt aus der Abstandsmatrix. Der optimale Pfad von oben links nach unten rechts geht über die Zellen  $(D_{31}, E_{30})$ ,  $(D_{32}, E_{30})$ ,  $(D_{33}, E_{31})$  und  $(D_{34}, E_{32})$ . Daher werden  $D_{31}$  und  $D_{32}$  zu  $E_{30}$  alignt. Manuell würde man aber ganz klar  $D_{32}$  und  $D_{33}$  zu  $E_{31}$  alignen. Der Aligner wählt dieses Alignment nicht, da der Abstand zwischen  $D_{32}$  und  $E_{31}$  größer ist als der Abstand zwischen  $D_{32}$  und  $E_{30}$ . Tabelle 7.17 zeigt die Abstände der Segmente. Hier wird das manuelle Alignment besser bewertet als das KoKS-Alignment.

### 7.5.3 Absatzalignment

Einige Teilkorpora konnten nicht verwendet werden, da die Zerlegung in korrekt alignte Absätze nicht gelang und die Verarbeitung des gesamten Dokuments zu viel Zeit in Anspruch genommen hätte. Wir haben daher versucht, die Absätze mittels Alignment einander zuzuordnen. Da der Aligner zu der Zeit noch

viele Fehler enthielt, schlug dies aber häufig fehl. Diese Versuche sollten jetzt wiederholt werden. Das Programm `align.sh` ist bereits dafür ausgelegt. (Allerdings wurde die Funktion noch nicht getestet.)

Das Trigrammabstandsmaß sollte für das Absatzalignment ausreichen, da i.d.R. jeder Absatz einen eigenen Themenschwerpunkt hat, der die auftretenden Eigennamen, Zahlen und Fachbegriffe eingrenzt. Vermutlich sind die Unterschiede zwischen den Absätzen deutlich größer als zwischen Sätzen des gleichen Absatzes. Das Trigrammabstandsmaß ist vermutlich für das Absatzalignment viel besser geeignet als für das Satzalignment.

#### 7.5.4 Evaluation

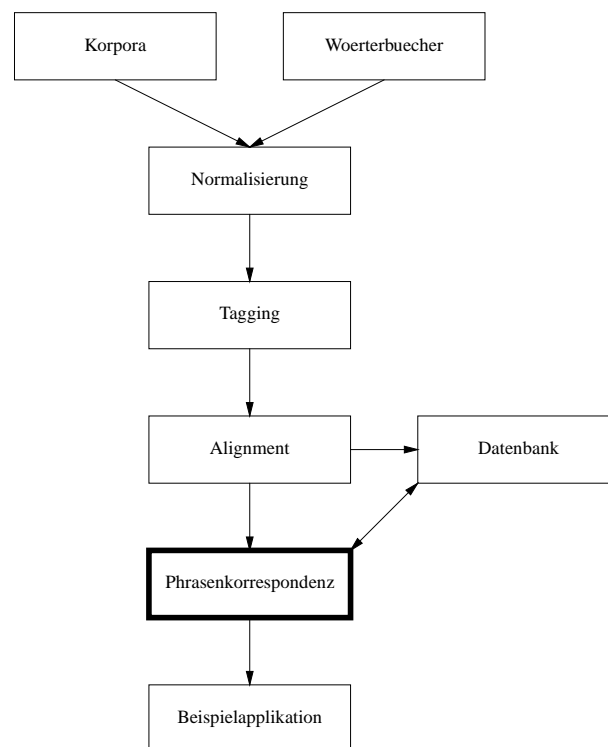
Bei aller Kritik und Verbesserungsvorschlägen sollte man beachten, dass das Alignment in den meisten Fällen korrekt ist. Dies liegt daran, dass die Sätze der verwendeten EU-Dokumente und Nachrichtentexte sehr häufig eins zu eins übersetzt werden. Um die Qualität der verschiedenen Alignmentverfahren beurteilen zu können müssen also viele Dokumente untersucht werden.

Für die automatische Auswertung stehen bereits in KoKS Werkzeuge zur Verfügung. Der Aligner `align_debug.sh` speichert eine textuelle Repräsentation des Alignment, die die gleichen Informationen wie die ersten drei Spalten von Tabelle 7.17 enthält. Die so erhaltenen `.align` Dateien können leicht mit UNIX-Tools mit manuell erstellten `.align` Dateien verglichen werden. Die in der vierten Spalte der Tabelle abgedruckten Abstände der Segmente können mit dem Programm `abstand.py` berechnet werden. (Siehe Anhang F.2.) Des Weiteren kann das Visualisierungs-Tool `Mavis` helfen. Die Repräsentation des Alignments als farbig in die Abstandsmatrix eingezeichneter Pfad erlaubt es, sehr leicht Abweichungen zwischen Alignments zu lokalisieren.

Für eine detaillierte Evaluation sollte ein spezielles Korpus zusammengestellt werden. Wenn möglich sollten Korpora verwendet werden, mit denen andere Aligner bereits untersucht wurden. Zum Beispiel wird in Kay und Röscheisen (1993) ausführlich auf die Ergebnisse eines Alignment eines Artikels aus der Zeitschrift 'Spektrum der Wissenschaft' eingegangen. Der Text ist recht frei übersetzt, sodass 3:2 Alignments und auch Überkreuzungen auftreten.

# Kapitel 8

## Phrasenkorrespondenz



### Inhaltsverzeichnis

---

8.1	Einleitung . . . . .	77
8.2	Chunk-Parsing mit Tagfolgen . . . . .	77
8.3	Die Phrasenzuordnung im Einzelnen . . . . .	80
8.4	Ergebnisse . . . . .	81
8.5	Ausblick . . . . .	82

---

## 8.1 Einleitung

Das Ziel von KoKS, wie es ja schon dem Projektnamen zu entnehmen ist, ist es, in parallelen Korpora Kollokationen zu finden (s. 1.1). Dazu müssen in den einander durch Alignment zugeordneten Sätzen

1. Phrasen identifiziert werden,
2. sie den Phrasen im zugeordneten anderssprachigen Satz zugeordnet werden, und
3. festgestellt werden, welche Phrasen Kollokationen sind.

Letzteres kann herausgefunden werden, indem man überprüft, ob die Phrasen kompositionelle Übersetzungen voneinander sind – sind sie es nicht, ist die Wahrscheinlichkeit recht hoch, dass es sich in einer Sprache um eine Kollokation handelt (vgl. Kapitel 2).

Zu diesem Zweck braucht man eine große Menge von alignten Korpora und wörterbuchbasierte Übersetzungsinformationen. In den vorigen Kapiteln wurde beschrieben, wie wir diese beschafft und organisiert haben.

### 8.1.1 Syntaktische Strukturen erkennen

Eine *Phrase* ist eine syntaktische Struktur, die aus mehreren Wörtern besteht. Um solche Strukturen zu finden, muss man im Grunde genommen Sätze parsen. Hat man einen Parser, der die Baumstruktur eines Satzes zurückliefert, so kann man die Phrasen leicht identifizieren. Allerdings ist diese Aufgabe sehr komplex: Viele Sätze haben mehrere mögliche Parsebäume, so dass man einen Parsebaum wählen muss, der dem Sinn des Satzes entspricht, was nicht einfach ist, da man u. U. auch die Semantik des Satzes und womöglich den Kontext in die Auswahl einbeziehen muss. Diese Aufgabe ist automatisch bisher nicht zufrieden stellend gelöst – zu wenige Sätze werden vollständig geparst, und es dauert lange.

Eine andere Möglichkeit ist es, die Sätze in ihre Bestandteile wie Nominalphrasen, Präpositionalphrasen, Verbalphrasen etc. aufgrund der Part-of-Speech-Tags der Wörter aufzuteilen. Dies wird z.B. beim *Chunking* oder *Chunk-Parsing* eines Satzes durchgeführt, wobei keine Parsebäume erzeugt werden, sondern der Satz in kleinere Teile, so genannte *Chunks*, zerlegt wird. Diese werden dann den Bestandteilen des entsprechenden Satzes in der anderen Sprache aufgrund von Wort-für-Wort-Übersetzungen zugewiesen.

### 8.1.2 Der Ansatz von KoKS

Wir haben in unserem Korpus Segmente, also Satzgruppen auf Englisch und Deutsch, die einander mit großer Wahrscheinlichkeit entsprechen. Außerdem haben wir ein Wörterbuch und somit die Möglichkeit, Wörter und Phrasen, die dort enthalten sind, zu finden. Was aber ist eine Phrase? Um aufwändiges Parsing zu vermeiden, haben wir einen rudimentären Chunk-Parsing-Ansatz gewählt. Wir haben Folgen von Tags, die oft hintereinander auftreten, gesammelt und kategorisiert. Mithilfe dieser Tagfolgen kann man ein Segment in Phrasen aufteilen. Kapitel 8.2 gibt darüber genauer Auskunft.

Wenn man von diesen Phrasen einer bestimmten Kategorie solche, die keine wörtliche Übersetzung im anderen Segment besitzen, Phrasen genau dieser Kategorie im anderen Segment zuweist, hat man einen Kollokationskandidaten. Manche Wörter, vor allem solche aus geschlossenen Wortklassen, sind bei diesem Verfahren zunächst irrelevant. Oft haben diese Wörter eine Entsprechung im anderen Segment, und außerdem sind sie nicht die Hauptbestandteile einer Kollokation. Bei unserem Chunk-Parsing haben wir die Chunks nur an Nomen, Verben und Adjektiven angelegt, weil diese die wichtigsten Bestandteile von Kollokationen sind. Das Phrasenalignment im Einzelnen wird in Kapitel 8.3 beschrieben.

## 8.2 Chunk-Parsing mit Tagfolgen

Das Kollokationsverständnis von KoKS (vgl. Kap. 2.3) ist syntaktisch orientiert: Nur Phrasen als Bestandteile von Sätzen oder Teile von Phrasen können Kollokationen sein. KoKS verfolgt mit dem *Chunk-Parsing* einen Ansatz, der keine tiefe strukturelle Syntaxanalyse vornimmt. Ziel ist es, syntaktisch zusammengehörende Wörter zu Chunks zusammenzufassen.

In welcher Relationen die einzelnen Chunks zueinander stehen, ist für KoKS nicht wichtig. Auch muss die Identifikation der Chunks nicht von vornherein korrekt und eindeutig sein: Ist ein möglicher Chunk

identifiziert, stehen die darin enthaltenen Wörter zunächst auch weiteren Chunks zur Verfügung. KoKS will ja Phrasen erkennen, um sie anschließend einem Gegenstück des zugehörigen anderssprachigen Satzes zuzuordnen. Dabei können für einen Satz auch mehrere Versuche gemacht werden, bis eine Phrasenzuordnung steht. Der Abgleich mit dem korrespondierenden Satz der anderen Sprache filtert die fehlerhaften Chunks aus. Die Kandidaten für Chunks können sich also überschneiden, die Chunks, die am Ende einem korrespondierenden Gegenstück im übersetzten Satz zugeordnet werden, sind aber disjunkt (Ausnahme: eine Phrase ist komplett in einer anderen enthalten, z.B. eine NP in einer PP).

Das zugegebenermaßen rudimentäre Chunk-Parsing in KoKS erfolgt dadurch, dass Folgen von Part-of-Speech-Tags einer möglichen Phrase zugeordnet werden. Welche Wortarten in einer Phrase, beispielsweise einer Nominalphrase (NP), aufeinander folgen, kann natürlich stark variieren, so dass für einen Phrasentyp viele verschiedene Hypothesen erstellt werden. Der folgende Abschnitt erläutert, wie diese Hypothesen generiert wurden. Dies erfolgte für Deutsch und Englisch auf unterschiedliche Weise.

### 8.2.1 Tagfolgen für Phrasen des Deutschen

Die Hypothesen für Tagfolgen in Phrasen des Deutschen wurden mit Hilfe des *cqp* (*Corpus Query Processor*) des IMS Stuttgart erstellt<sup>1</sup>. *cqp* ist ein Tool, mit dem auf annotierten Korpora sehr flexibel Anfragen gestellt werden können, z.B. nach Häufigkeit und Kontext einer bestimmten Wortform, eines Lemmas, einer Wortart und dergleichen mehr. Die Annotation der zu Grunde liegenden Korpora umfasst neben Part-of-Speech-Tags auch die Markierung von Chunks.

Die Chunk-Parsing-Funktionalität von *cqp* steht allerdings nicht als unabhängiges Tool zur Verfügung, sondern das Chunking wird bereits erledigt, wenn ein Korpus für *cqp* aufbereitet wird (das annotierte Korpus wird in ein proprietäres Binärformat überführt). Das Korpus von KoKS ist dagegen in eine Datenbank eingetragen. Ein Chunk-Parsing mit *cqp* auf dem KoKS-Korpus direkt war also nicht durchzuführen.

Stattdessen wurden auf einem anderen Korpus (der 93er Jahrgang der *Frankfurter Rundschau*) sämtliche zur Verfügung stehenden Chunks extrahiert und aufgesammelt. Dabei waren nicht die Wortformen, sondern nur die PoS-Tags von Interesse; mit *cqp* ist dies aber ohne weiteres möglich. Die Ergebnisse wurden in eine Datei umgelenkt und anschließend mit UNIX-Tools (vornehmlich *sort* und *uniq -c*) weiterverarbeitet, so dass eine (lange) Liste von Tagfolgen mit der jeweiligen Häufigkeit im FR-Korpus entstand. Auf diese Weise ließen sich Tagfolgen für Nominalphrasen (NPs), Präpositionalphrasen (PPs) und Verbkomplexe (VCs) aufstellen. Bei letzterem handelt es sich um Komplexe, in denen Verben (u.U. gemeinsam mit Partikeln) zusammen ein Prädikat bilden, das auch aus mehreren Wörtern bestehen kann (z.B. „zu tun gedenken“).

Für KoKS sind aber auch Verbalphrasen (VPs) wichtig. Um auch hierfür Tagfolgen erstellen zu können, wurden alle theoretisch möglichen Kombinationen aus VC, NP und PP extrahiert (z.B. *VC NP PP* in *[Der Mann]<sub>NP</sub> [[schlägt]<sub>VC</sub> [den Hund]<sub>NP</sub> [mit dem Knochen]<sub>PP</sub>]<sub>VP</sub>*). So entstanden Listen mit Hypothesen für Tagfolgen in VPs, NPs und PP, von denen die seltenen ignoriert wurden.

### 8.2.2 Tagfolgen für Phrasen des Englischen

Für das Englische stand kein aufbereitetes Korpus in *cqp* zur Verfügung; das Verfahren zur Erstellung deutscher Tagfolgen ließ sich also nicht für Englisch nutzen. Die Hypothesen wurden folglich „von Hand“ erstellt. Dafür wurden UNIX-Tools (*grep*, *sort*, *uniq* u.a.) sowie kleine Perl-Skripte eingesetzt. Dies war möglich, da der Inhalt des englischen Korpusteils nicht nur in der Datenbank, sondern auch in Form von Textdateien vorlag.

Das Vorgehen dabei war folgendes: Die Dateien mit den englischen Texten wurden getaggt, der Output in einer Datei gesammelt. Auf dieser wurde dann nach Tagfolgen, die bestimmte Kriterien erfüllen, gesucht. Es handelt sich bei diesem Vorgehen nicht unbedingt um ein linguistisch fundiertes Verfahren; da für KoKS vor allem ein hoher *Recall* (keine relevante Tagfolge soll übersehen werden) und weniger die *Precision* (nur relevante Tagfolgen sollen gefunden werden) wichtig ist, stört das zunächst nicht. Die Idee ist:

1. Eine **NP** ist eine Folge von Tags von einem Artikel bis zu einem oder mehreren, evtl. durch „of“ verbundenen Substantiven. Hinzu kommen die gleichen Folgen ohne den führenden Artikel.

<sup>1</sup>s. <http://www.ims.uni-stuttgart.de/projekte/CorpusWorkbench/>; ein Online-Handbuch zu *cqp* findet sich unter <http://www.ims.uni-stuttgart.de/projekte/CorpusWorkbench/CQPUserManual/HTML/>

DT	NN	VBZ	IN	NN	VBD	VCN	RP
The	school	's	out	party	was	called	off.
ART	NN	APPRART		NN	VVFIN	APPRART	NN
Die	Fete	zum		Ferienbeginn	fiel	ins	Wasser.

Abbildung 8.1: Ausgangssegment

DT	NN	VBZ	IN	NN	VBD	VCN	RP
<u>The</u>	school	's	<u>out</u>	party	was	called	<u>off.</u>
ART	NN	APPRART		NN	VVFIN	APPRART	NN
<u>Die</u>	Fete	<u>zum</u>		Ferienbeginn	fiel	<u>ins</u>	Wasser.

Abbildung 8.2: Schritt 1: Wörter mit irrelevanten Tags streichen

2. Eine **PP** ist eine Präposition mit anschließender NP. Auch ein „to“ kann als Präposition fungieren; es wird vom Tagger wegen der vielfältigen Möglichkeiten aber als „TO“ getaggt, so dass eine Folge von „TO“ plus NP auch als PP identifiziert wird.
3. Ein **VC** (Verbkomplex) ist eine Folge von Verben, evtl. verbunden durch ein „to“
4. Eine **VP** ist eine VC, eine Folge von VC und NP, eine Folge von VC, NP und PP oder u.U. eine Folge von VC und PP.

Die Punkte 1 bis 3 lassen sich recht leicht extrahieren. Die Generierung von Verbalphrasen erfolgte durch „Ausmultiplizieren“: An jede Tagfolge, die ein VC bildet, werden alle Tagfolgen, die eine NP bilden können, angehängt usw. Dadurch entstehen sehr viele Kandidaten, u.a. auch viele Tagfolgen, die gar nicht (im Korpus) auftauchen. Für jeden VP-Kandidaten wurde also sein tatsächliches Vorkommen im englischen Teil des KoKS-Korpus gezählt; selten bis gar nicht auftretende Tagfolgen wurden ignoriert.

DT	NN	VBZ	IN	NN	VBD	VCN	RP
<u>The</u>	school	's	<u>out</u>	<u>party</u>	was	called	<u>off.</u>
ART	NN	APPRART		NN	VVFIN	APPRART	NN
<u>Die</u>	<u>Fete</u>	<u>zum</u>		Ferienbeginn	fiel	<u>ins</u>	Wasser.

Abbildung 8.3: Schritt 2: Übersetzungen markieren

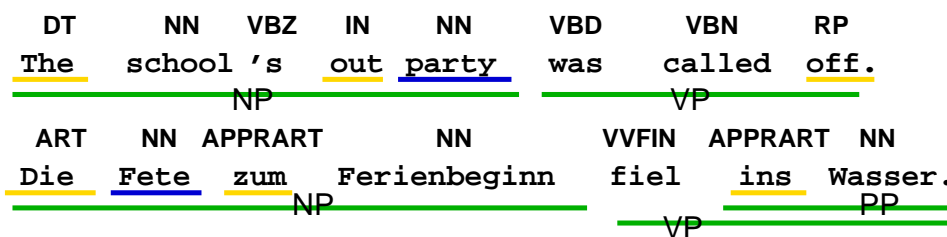


Abbildung 8.4: Schritt 3: Mithilfe von Tagfolgen Phrasen erkennen

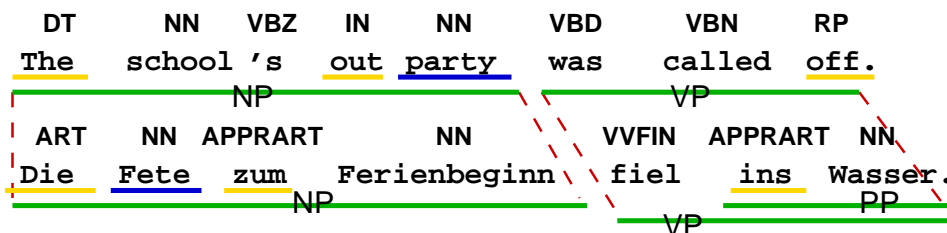


Abbildung 8.5: Schritt 4: Tagfolgen einer Kategorie einander zuordnen

### 8.3 Die Phrasenzuordnung im Einzelnen

Wie genau funktioniert die Phrasenzuordnung in KoKS? An einem Beispielsegment (siehe Abbildung 8.1) soll dies erläutert werden.

1. Man markiert Wörter als unwichtig, deren Tags als irrelevant befunden worden sind, also solche aus geschlossenen Wortklassen, oder solche, die der Tagger nicht erkennt. (Abbildung 8.2)
2. Wörter, deren Übersetzung mithilfe des Wörterbuchs im Satz der anderen Sprache gefunden werden kann, werden markiert, um später nicht Phrasen einander zuzuordnen, die schon im Wörterbuch stehen (Abbildung 8.3)
3. Es wird versucht, die gesammelten Tagfolgen an die nicht als unwichtig markierten Wörter anzulegen. Es müssen also alle Tags der Wörter um ein bestimmtes Wort mit einer Tagfolge übereinstimmen, dann hat man eine Phrase. Die Kategorie der Tagfolge, also Nominalphrase, Verbalphrase etc., wird notiert. (Abbildung 8.4)
4. Für jede so gefundene Phrase wird überprüft, ob es eine Phrase der gleichen Kategorie im entsprechenden anderssprachigen Segment gibt. Ist dies der Fall, so werden sie einander als mögliche Übersetzungen zugeordnet. Weiter bearbeitet werden aber nur Phrasenpaare, bei denen zumindest eine der Phrasen nicht an einem Wort angelagert wurde, das in Schritt 2 als übersetzt markiert wurde, also wo keine schon durch das Wörterbuch bekannte Übersetzungen vorliegen. Die Phrasen „The school's out party“ und „Die Fete zum Ferienbeginn“ werden allerdings weiter bearbeitet, da der Chunk nicht nur am Wort „Party“ bzw. „Fete“ angelagert wurde, sondern auch am Wort „school“ bzw. „Ferienbeginn“. (Abbildung 8.5)
5. Alle diese so zugeordneten Phrasen werden in die Datenbank eingetragen und das Segment, aus dem sie extrahiert wurden, als Belegsegment notiert, das den Kontext veranschaulicht.

Beim Eintragen in die Datenbank wird mitgezählt, wie oft eine bestimmte Phrase gefunden wurde. Wenn man genügend Segmente durchgearbeitet hat, kann man aufgrund der zugeordneten Phrasen eine Untergrenze der Häufigkeit festlegen, ab der man die Phrasen als sicheres Paar bezeichnen kann. Es mag zwar viele nicht-kollokative Phrasen geben, die so zugeordnet werden, bei einem genügend großen Korpus sollte es aber keine Häufungen bei einer bestimmten Phrase geben.

Belege pro Phrase	Häufigkeit
1	73427
2	275
3	95
4	31
5	19
6	6
7	10
8	4
9	5
10	4
11	1
12	2
13	1
14	2
18	1
19	1
20	1
24	1
42	1
44	1
47	1
80	1
100	1
101	1
161	1

Tabelle 8.1: Häufigkeit der Belege pro Phrase

### 8.3.1 Tools

Das Skript `phrasealign.py` erwartet eine Liste von Segment-Ids aus der Datenbank und versucht dann, mithilfe des erläuterten Vorgangs Phrasen zuzuordnen und trägt die erkannte Phrasen in die Datenbank ein. Näheres siehe Kapitel G.

## 8.4 Ergebnisse

Von Mitte September bis zum 14. November hat unser System 3119 Segmente durchsucht und dabei 73885 Kandidaten für Phrasenpaare gefunden. Davon wurden 466 Phrasen mehr als einmal gefunden. Die genaue Verteilung der Phrasen ist in Tabelle 8.1 zu sehen. In Tabelle 8.2 folgen einige der erkannten Beispiele.

Man sieht an den erkannten Phrasen, welche Art von Texten wir verwendet haben. Da die Phrasenpaare „die europäische Kommission“ und „the European Commission“ oft im Korpus vorkommen, tauchen auch alle möglichen Kombinationen oft auf. Wenn man viele verschiedenen Korpora nach diesem Verfahren bearbeitet hat, fallen vermutlich auch die vielen Kombinationen von „Kommission“ weg. Leider wurden bisher nur drei Verbkombinationen gefunden: „has been“ – „wurde“, „beruht“ – „is based“ und „ins Leben gerufen“ – „was launched“, die einzige wirkliche Kollokation. Auch hätte man beim Eintragen sich auf Lemmata beschränken können und, wenn man Phrasen mit Artikeln gefunden hat, die Eintragung ohne Artikel löschen können. Auch muss die Qualität der gefundenen Phrasen erhöht werden, indem man darauf besteht, dass eine der beiden Phrasen länger ist als ein Wort - ansonsten ist es eben keine Kollokation.

Deutsche Phrase	Englische Phrase	Häufigkeit
Kommission	The Commission	101
Die Kommission	Commission	100
Die Kommission	The Commission	80
die Kommission	Commission	47
Kommission	the Commission	44
die Kommission	the Commission	42
Mitgliedstaaten	Member States	20
der Kommission	Commission	19
Die Europäische Kommission	European Commission	14
Die Europäische Kommission	The European Commission	13
der Kommission	the Commission	12
die Kommission	The Commission	12
Europäische Kommission	European Commission	11
Die Europäische Kommission	Commission	10
Die Europäische Kommission	European	10
Die Europäische Kommission	The European	10
wurde	has been	9
...	...	...
ins Leben gerufen	was launched	3
der Grundsatz	the principle	2
Schwarzmeer-Region	Black Sea region	2
Vertragsverletzungsverfahren	infringement procedures	2
beruht	is based	2
...	...	...

Tabelle 8.2: Beispiele für erkannte Phrasen

## 8.5 Ausblick

Die Qualität der Ergebnisse kann zum einen durch Verbesserung der Quellen und zum anderen durch Verbesserung unseres Algorithmus erhöht werden. Hat man viele Korpora, die gut align sind, und Wörterbücher, die viele Wörter abdecken, so sollten die Ergebnisse auch mit dem jetzigen Algorithmus schon besser werden – auch die geringe Zahl der bis jetzt bearbeiteten Sätze ist ein Problem. Den Algorithmus kann man verbessern, indem man z. B. nur auf einer Seite Einzelwörter erlaubt und nur Lemmata einträgt, soweit es geht. Auch sollten die eingetragenen lemmatisierten Phrasen sortiert werden, damit man an sich identische Phrasen, bei denen sich nur die Wortstellung unterscheidet, nicht als zwei verschiedene Phrasen einträgt. Mit unserem Algorithmus würden “ist ins Wasser gefallen” und “fiel ins Wasser” als verschiedene Phrasen erkannt.

Zu Precision und Recall, also fälschlich erkannten Phrasenpaaren bzw. übersehenen Phrasenpaaren, haben wir keine Zahlen. Anhand der gefundenen Phrasen lässt sich aber abschätzen, dass die Precision annehmbar ist - wir haben kaum Phrasenpaare gefunden, die einander gar nicht entsprechen. Der Recall wird schlechter sein, da wir bisher nur eine Verb-Nomen-Konstruktion gefunden haben, in den Texten sollten aber insgesamt mehr vorkommen. Es ist also noch einiges Tuning am Ansatz und am Algorithmus nötig.

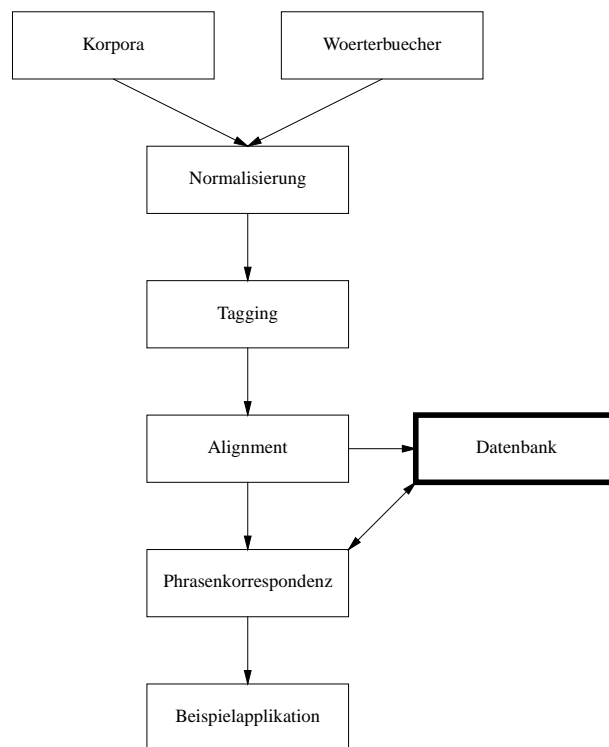
Insbesondere bei der Gewinnung der Tagfolgen drängt sich eine Frage zum geschilderten Verfahren auf: Leidet die Güte der Tagfolgen-Hypothesen nicht unter der schieren Menge? Für das Deutsche beispielsweise wurden 524 verschiedene Hypothesen genutzt, allesamt Tagfolgen, die mindestens 100 Mal im DE-News-Korpus vorkommen. Mögen die Heuristiken zur Gewinnung von Tagfolgen für VCs, NPs und PPs noch linguistisch plausibel sein, so wird doch dieser Rahmen durch das „Ausmultiplizieren“ von Tagfolgen verlassen. Die Hypothesen für VPs enthalten vermutlich viele Tagfolgen, die unter keinen Umständen eine VP bilden können. Insbesondere die flexible Wortstellung im Deutschen – zu erwähnen ist hier auch

die unterschiedliche Position des Verbs an zweiter oder letzter Stelle des Satzes – sorgen für eine schwer überschaubare Kandidatenmenge.

Eine Verbesserung der Phrasenzuordnung kann also dadurch erzielt werden, dass die bisherigen Tagfolgen-Hypothesen linguistisch fundierter überprüft werden. Sinnvoll wäre zudem eine Chunk-Parsing-Komponente, die direkt auf dem zu analysierenden Satz arbeitet, anstatt eine (Un-)Menge an Kandidaten darauf loszulassen.

# Kapitel 9

## Datenbank



### Inhaltsverzeichnis

---

9.1	Motivation	85
9.2	Konzepte	85
9.3	Unser Vorgehen	85
9.4	Design der KoKS-Datenbank	85
9.5	Tools	87
9.6	Ausblick	87

---

## 9.1 Motivation

Nach der Vorverarbeitung sind die Korpora alle in einem einheitlichen Format. Jetzt gilt es, den Zugriff darauf einfach und schnell zu ermöglichen.

## 9.2 Konzepte

Um große Textmengen zu verarbeiten, gibt es mehrere Möglichkeiten. Man kann nur mit den Dateien arbeiten, sie alle in eine *Datenbank* schreiben, oder eine Mischung von beidem machen. Will man effizient auf Texte zugreifen, programmiert man womöglich selber eine Datenbank nach, deshalb sollte man sich frühzeitig überlegen, welche Zugriffe man braucht.

## 9.3 Unser Vorgehen

Wir haben uns frühzeitig dafür entschieden, eine Datenbank zu benutzen. Letztendlich haben wir jedoch darüber hinaus einige Mechanismen selber nachprogrammiert, nämlich den Anfrage-Server mit einem Cache und spezialisierten Anfragen.

Die vorverarbeiteten Korpora werden in eine Datenbank eingetragen, um möglichst effizient darauf zugreifen zu können. Nach einigen Schnelligkeitstests haben wir uns für *MySQL* entschieden, weil es auf Transaktionen verzichtet und deshalb einen Geschwindigkeitsvorteil bietet. Jedoch gibt es Einschränkungen - MySQL bietet weder Unions noch Subselects, und so mussten wir in unseren Programmen dies selbst implementieren.

In der Datenbank werden die einzelnen Wörter, von uns als Tokens bezeichnet, aufgrund ihrer Grundform und ihres Tags unterschieden. Es gibt die Möglichkeit, den Sätzen eine Parsequalität zuzuordnen, und den Tokens eine Tagqualität, dieses wurde jedoch nicht genutzt. Auch Satzzeichen werden als Tokens angesehen und eingetragen, aus programmtechnischen Gründen wurden jedoch alle Anführungszeichen ` ' " durch # ersetzt.

## 9.4 Design der KoKS-Datenbank

### 9.4.1 Tabellen

In der KoKS-Datenbank gibt es die folgenden Tabellen:

**Autoren:** Autorid, Name

**Herkunft:** Herkunftsid, Name, Autorid, Jahr, Quellenid, Statusid

**Quellen:** Quellenid, Name, Verlag, Rechte, Bemerkungen

**Tokens:** Tokenid, Name, Reverse, Grundformid, Tagid, Tagqualitaet, Tagsetid

**Grundformen:** Grundformid, Name, Tagid, Tagsetid

**Sätze:** Satzid, Herkunftsid, Parsequalität, Segmentqual, Prev, Succ, Sprachenid

**Tagset\_ims\_de:** Tagid, Name

**weitere Tagsets**

**Tagsets:** Tagsetid, Name, Sprachenid

**Sprachen:** Sprachenid, Name

**Status:** Statusid, Name

**map\_s\_w:** Mapid, Satzid, Tokenid, Segmentnr

**Phrase.count:** Pc\_id, Segmentnr, Zähler

**Phrase.examples:** Pe\_id, Pc\_id, Segmentnr

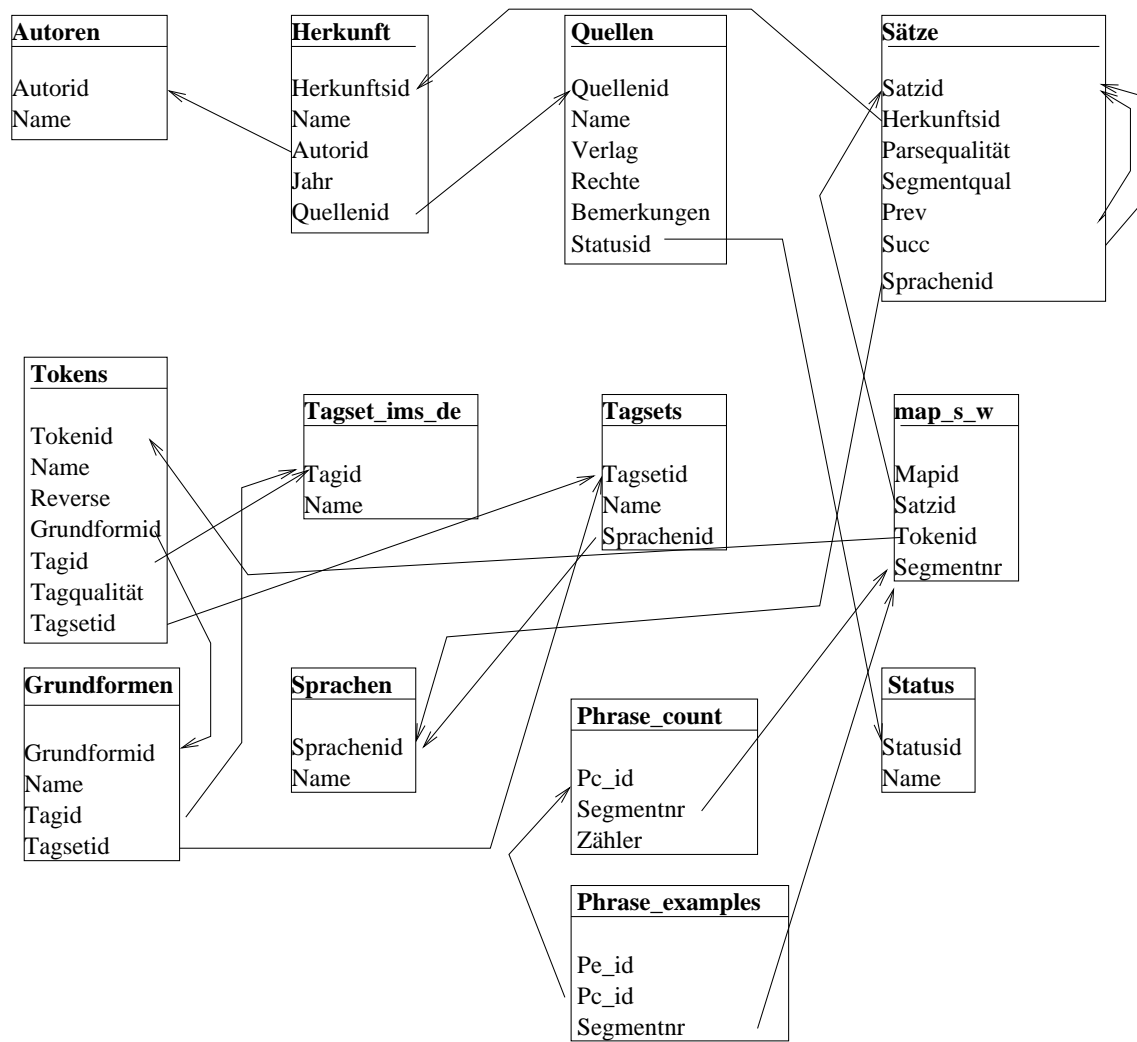


Abbildung 9.1: Das Datenbankdesign und die Verzeigerung veranschaulicht

### 9.4.2 Beschreibung des Designs

Die Relationen der Tabellen sind in Abbildung 9.4 zu sehen.

Die Herkunft eines Textes wird in der Tabelle *Herkunft* gespeichert. Diese hat einen Namen, eine Jahresangabe, einen Zeiger auf einen Autor und auf eine Quelle. Jeder Autor wird mit Namen gespeichert, bei einer Quelle werden der Verlag, die Rechte zur Weitergabe und etwaige Bemerkungen aufgeführt. Jeder Satz verweist auf eine solche Herkunftsstruktur.

Jedes Wort bzw. Token wird in der Tabelle *Tokens* mit einem Zeiger auf das Tag und das Tagset sowie einem Zeiger auf die Grundform gespeichert. Das Wort wird sowohl vorwärts als auch rückwärts abgelegt, um eine Suche mit *like* auch für Komposita zu ermöglichen (z.B. alle Wörter, die auf *-politik* enden: *where reverse = 'kitilop'*).

Sätze haben außer einem Zeiger auf die Herkunft und auf den vorherigen bzw. nachfolgenden Satz auch einen Zeiger auf ihre Sprache, und die Parsequalität.<sup>1</sup>

Die Zuordnung von Sätzen zu Wörtern erfolgt in der Tabelle *map\_s\_w*, wo jedem Segment die Sätze mit ihren Wörtern zugeordnet werden. Ein Segment ist in diesem Zusammenhang eine Menge von Sätzen im Deutschen und Englischen, die einander vom Aligner als Übersetzungen zugeordnet worden sind.

Die Tabellen *Tagsets*, *Sprachen*, *Status*, und die Tagset-Tabellen wie *Tagset\_ims\_de*, *Tagset\_ims\_en* etc. ermöglichen die Zuordnung von Tagsets zu Tokens, Sprachen zu Sätzen bzw. Status der Verarbeitung zu Texten.

Die Tabellen *Phrase\_examples* und *Phrase\_count* werden durch die Phrasenzuordnung gefüllt. In die Tabelle *Phrase\_count* wird ein Zeiger auf eine bestimmte, im Rahmen der Phrasenzuordnung erkannte Phrase sowie ein Zähler, wie oft diese Phrase erkannt wurde, eingetragen. Die Tabelle *Phrase\_examples* enthält Zeiger auf Beispielsegmente, in denen die in *Phrase\_count* gespeicherten Phrasen vorkommen.

## 9.5 Tools

Es gibt Skripte, um die Datenbank anzulegen und zu löschen, siehe das Kapitel O. Ein Python-Wrapper ermöglicht den Zugriff auf verschiedene Datenbanken.

Außerdem gibt es den Anfrage-Server, der auf die Datenbank zugreift, die Ergebnisse aber in einem Cache speichert, um den Zugriff zu beschleunigen. Ist die Datenbank eingerichtet und sind Korpora eingetragen, kann man den Server mit `nohup demo-qdb.py -s 2>&1 >server.log` im Verzeichnis `align` starten. Anfragen kann man am besten mit dem gleichen Tool im Client-Mode vornehmen: `demo-qdb.py -c`. Die genaue Syntax der Anfragen wird im Kapitel H beschrieben.

## 9.6 Ausblick

Das Datenbankdesign wurde schon recht früh entwickelt, somit waren wir darauf festgelegt. Um mit den Problemen des Datenbankdesigns bzw. der Geschwindigkeit umzugehen, haben wir einiges selbst programmiert, was vielleicht ohne die Verwendung von Datenbanken oder bei der Mischung von Datenbanken und Zeigern in die Dateien hätte wegfallen können. Bei großen Mengen von fortlaufendem Text ist eine Datenbank vielleicht nicht die beste Lösung.

---

<sup>1</sup>Die Parsequalität wird von KoKS weder bestimmt noch verwendet. Sie wird von LogoTax (Ludewig, 2001) über den Anfrageserver (Kap. H.3) mit Daten gefüllt.

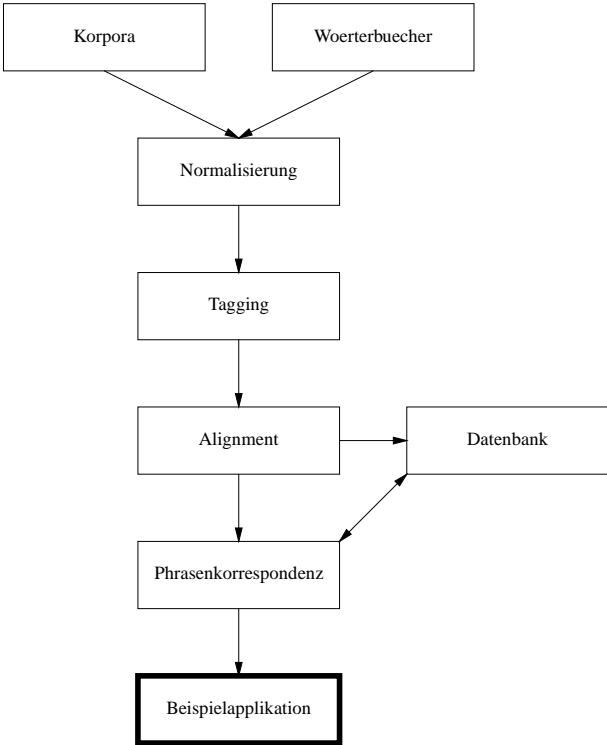
**Teil III**

**Anwendung**



# Kapitel 10

## Beispielapplikation



### Inhaltsverzeichnis

---

10.1 Motivation . . . . .	91
10.2 Technik und Benutzung (Anwendungsszenario) . . . . .	91
10.3 Middleware . . . . .	93
10.4 Ausblick . . . . .	94
10.5 Resume . . . . .	94

---

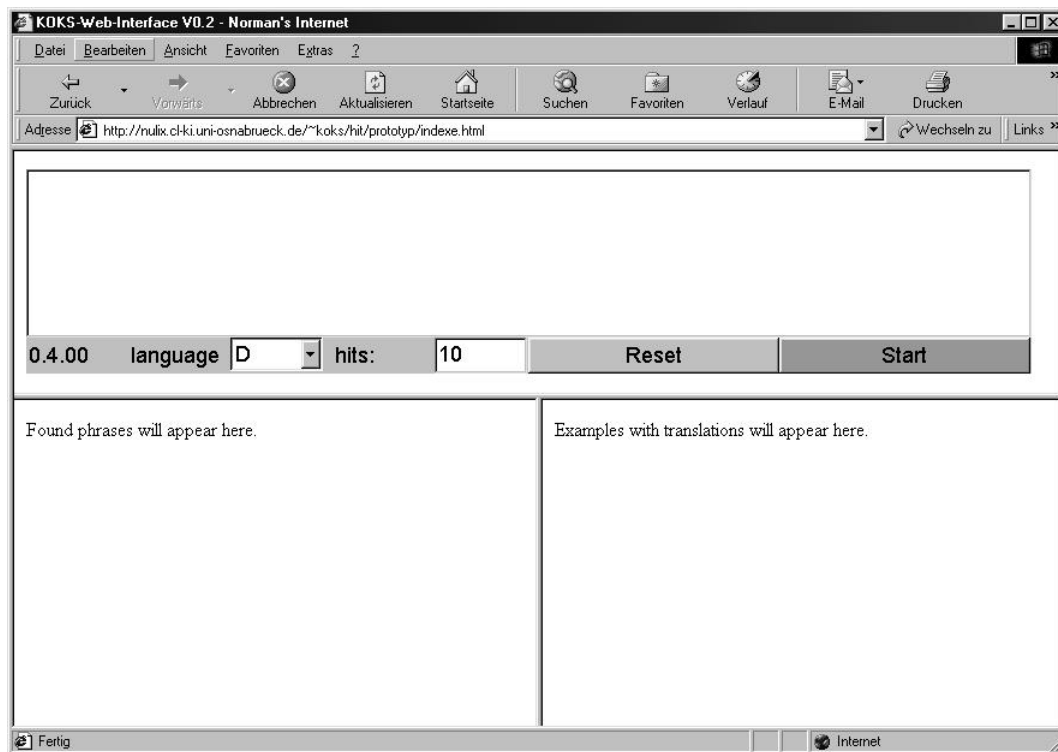


Abbildung 10.1: Die leere Demoapplikation.

## 10.1 Motivation

Das KoKS-System ist wie ein Eisberg der im Meer schwimmt - viele Daten in der Datenbank, Programme die den Korpus bearbeiten, Abfrage- und Analysewerkzeuge u.a. Dies sind alles mächtige Komponenten die im Hintergrund arbeiten und vom Benutzer nicht gesehen werden. Um die Leistungsfähigkeit und Möglichkeiten unseres Systems für einen Nutzer wahrnehmbar demonstrieren zu können, wurde eine Demo-Applikation (Prototyp) erstellt. Diese ist sozusagen die Spitze des Eisberges, also der sichtbare Teil.

Unsere Applikation ist im CALL-Bereich (s. Abschnitt 11.1) angesiedelt. Um weltweite Erreichbarkeit zu ermöglichen, wurde eine Internetanwendung<sup>1</sup> entwickelt.

## 10.2 Technik und Benutzung (Anwendungsszenario)

Es kann ein vom Anwender nicht verstandener (nicht Wort für Wort mit einem Wörterbuch übersetzbarer) Satz eingegeben werden, mit oder ohne Kontextsätzen. Danach wird in diesem Satz eines der nicht verstandenen Wörter markiert.

Die Applikation besteht aus einem Java-Applet und zwei Ruby-Scripten. Für die Texteingabe und Selektierung wird das Java-Applet (s. Abschnitt P.1.1: KoksAppe.java) benutzt. Pläne einer reinen Java-Applikation wurden aus Performancegründen verworfen, auch wurde wegen der Browser-Kompatibilität auf Java-Swing verzichtet.

Das Java-Applet extrahiert diesen Satz und das markierte Wort und schickt einen HTTP-Request:

```
http://nunix.cl-ki.uni-osnabrueck.de/~koks/demo/cgi-bin/hitle.rb
?port=29295&phrases=Er hat <i> nichts </i> zu sagen .
```

<sup>1</sup><http://nunix.cl-ki.uni-osnabrueck.de/~koks/eurocall/demo/indexe.html>

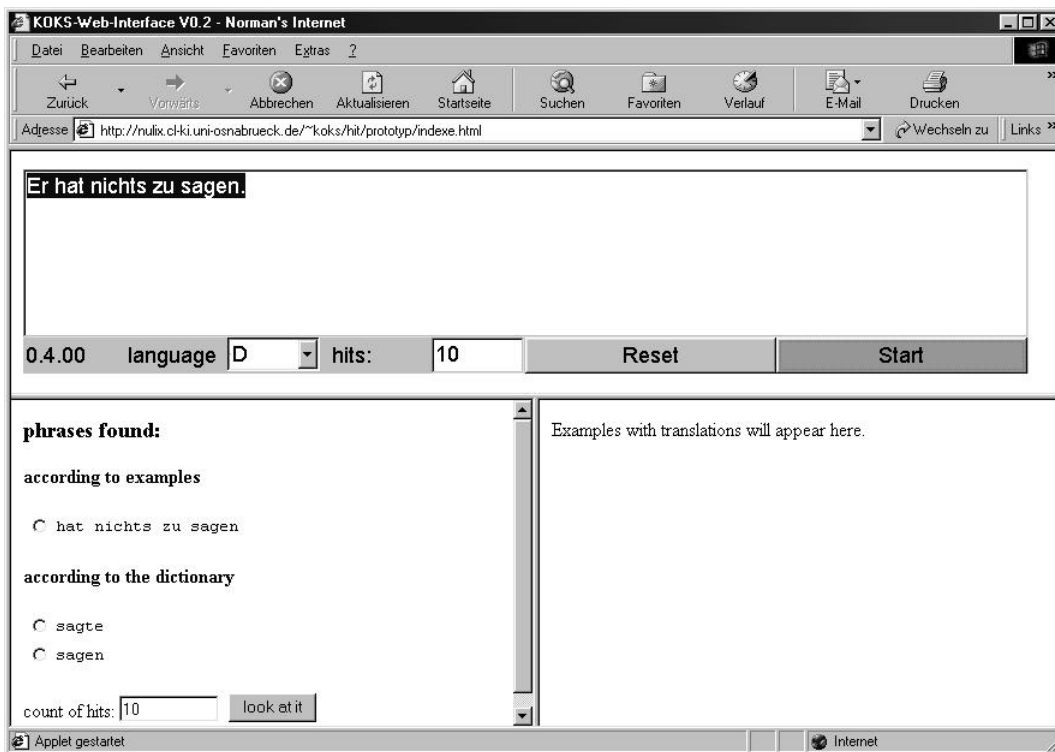


Abbildung 10.2: Das Wort "hat" wurde markiert.

```
&max=10&t=t:segment&s=s:text&applet=true&lang=de&verbose=1
```

Parameter	Erklärung
port=29295	Port für Datenbank-Abfrageserver
phrase=Er hat <i> nichts </i> zu sagen	extrahierter Satz, das <i>-Tag kennzeichnet ausgewähltes Wort
max=10	Anzahl der anzuzeigenden Treffer
t=t:segment	Suche in Segmenten (satzübergreifend)
s=s:text	Ausgabe im Textformat
applet=true	Anfrage kommt vom
lang=de	Sprache des eingegebenen Textes ist deutsch
verbose=1	Applet protokolliert den geschickten Request

an ein Ruby-Skript (s. Abschnitt P.1.2: hit1.rb)

Zuerst wird der Satz analysiert, danach werden alle im Satz möglichen Wortkombinationen gebildet, die das markierte Wort enthalten. Die so entstandenen Phrasen werden mit den im System vorhandenen verglichen, es werden bekannte Phrasen gesucht und danach dem Anwender präsentiert (im Frame unten links). Hierbei wird gekennzeichnet, ob die Phrasen aus den Wörterbüchern kommen oder vom KOKS-System selbst erkannt wurden.

Der Anwender markiert nun im linken Frame eine Phrase (siehe Abbildung 10.2), zu welcher das System Übersetzungen bereitstellen soll. Dafür wird ein weiteres Ruby-Skript (s. Abschnitt P.1.3: hit2.rb) herangezogen.

Als Ergebnis werden Beispielvorkommen der extrahierten Phrase aus unseren im System vorhandenen parallelen Korpora angezeigt (siehe Abbildung 10.3), natürlich mit den jeweiligen Übersetzungen.

Im Frontend können zusätzlich einige Einstellungen vorgenommen werden, welche die angezeigte Trefferanzahl beeinflussen.

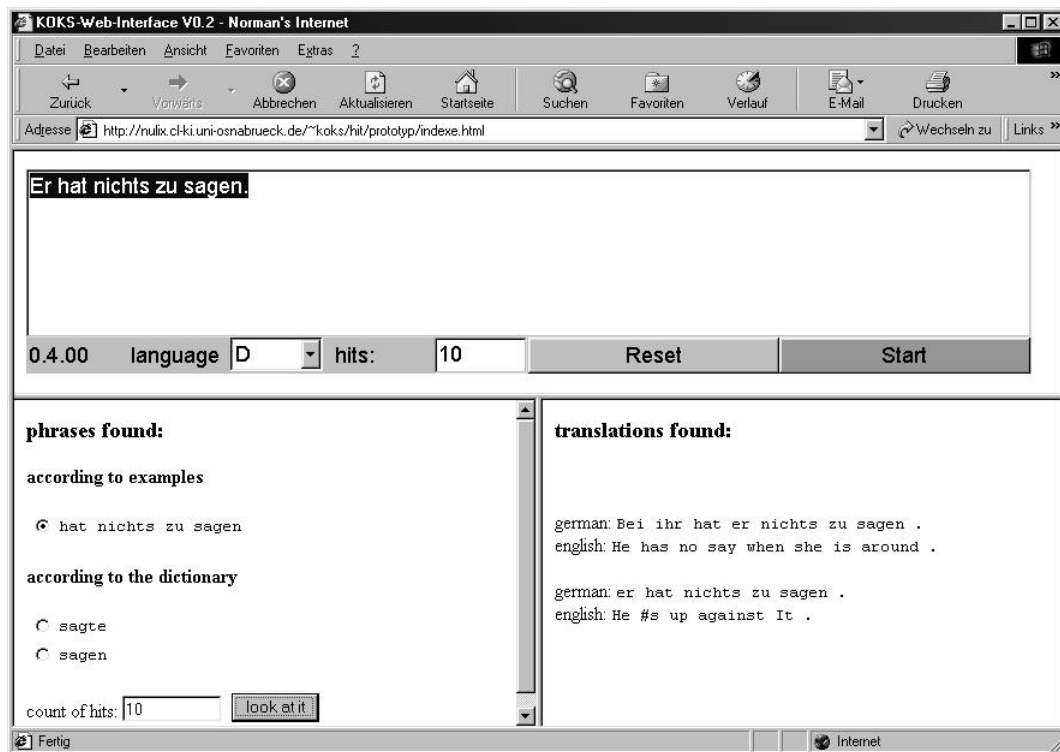


Abbildung 10.3: Es wurde links unten die gefundene Kollokation "hat nichts zu sagen" ausgewählt.

### 10.3 Middleware

Das Applet kommuniziert nicht direkt mit der Datenbank, sondern über HTTP-Requests mit einem CGI-Skript. Diese "Middleware" hat zwei Aufgaben: Zum einen wertet sie die Anfrage des Applets aus und gibt sie an den Abfrage-Server weiter. Zum anderen schickt sie die Antworten vom Abfrage-Server in formatierter Form an den Browser zurück. Ebenso erfolgt die Bearbeitung einer Anfrage, wenn der Benutzer im Browser eine Phrase zur genaueren Betrachtung auswählt. Zum Ablauf der Kommunikation zwischen Browser, Middleware und DB-Server siehe Abbildung 10.4. Der Abfrage-Server wird in Abschnitt H.3.1 vorgestellt.

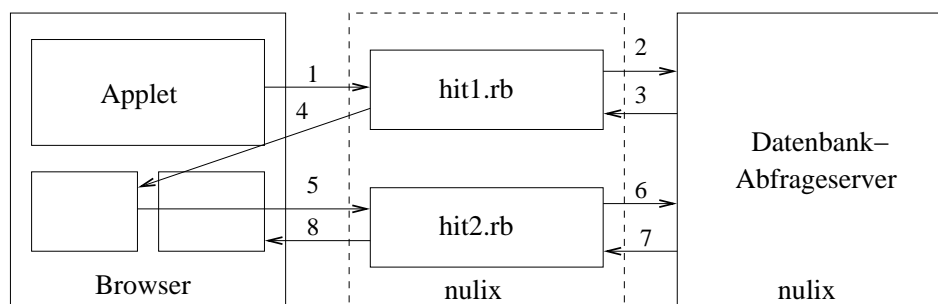


Abbildung 10.4: Kommunikation

## 10.4 Ausblick

Das KoKS-System kann in weiteren Applikationen eingesetzt werden, z.B.:

- als Point-To-Klick-Anwendung, ähnlich dem Babylon-Translator <sup>2</sup>
- als PlugIn für Textverarbeitungsprogramme
- als Modul für Übersetzungsprogramme
- in allen Programmen, die authentische Verwendungsbeispiele präsentieren wollen

## 10.5 Resume

Die hier beschriebene prototypische Anwendung des KoKS-Systems unterstützt den Nutzer beim Erkennen und Lernen von kollokationsartigen Multiwortkonstruktionen. LogoTax, ein in unserem Institut entwickeltes CALL-Programm <sup>3</sup> verwendet schon heute erfolgreich die KoKS-Technologie.

---

<sup>2</sup><http://www.babylon.com>

<sup>3</sup><http://www.cl-ki.uos/~LogoTax>

# Kapitel 11

## Call Kontext

### Inhaltsverzeichnis

---

11.1 Motivation . . . . .	95
11.2 Das KoKS-System und CALL . . . . .	95
11.3 Ausblick . . . . .	96

---

### 11.1 Motivation

#### 11.1.1 Was ist CALL?

Im Rahmen von *e-learning* steht CALL für Computer Assisted/Aided Language Learning". Es hat insbesondere die Unterstützung beim Sprachenlernen zum Ziel, soll Lehrer aber nicht ersetzen.

Kollokationen und Multiwortkonstruktionen im Kollokationsumfeld sind für Nicht-Muttersprachler schwer zu erlernen, da sie oftmals nicht als solche erkannt werden und nicht Wort für Wort mit einem Wörterbuch übersetzbar bzw. darin schwer zu finden sind.

#### 11.1.2 Konzepte und Anwendungsbereiche

Laut Rüschoff&Wolf 99 gibt es zwei große Gruppen von CALL-Programmen, tutorielle und nichttutorielle.

Tutorielle Programme arbeiten oft auf der Basis von einfachen Masken und richtig/falsch-Vergleichen. Sie stellen Aufgaben mit Lösungshinweisen und bewerten die Qualität der Antworten. Leider bieten sie meist keine Eingabeanalyse, widersprechen oft den Gestaltungsprinzipien des modernen Fremdsprachenunterrichts und sind für die Realisierung der Lernziele "Kommunikationsfähigkeit", "Sprach- und Lernbewusstsein" nicht geeignet.

Nichttutorielle Programme haben oft einen eher explorativen Charakter. Es sind Übungen zur Textkonstruktion und -rekonstruktion eingebaut. Diese Programme werden auch als Werkzeuge oder Ressourcen bezeichnet.

Das KoKS-System ist in den letzteren Bereich einzuordnen, denn es ist ein Werkzeug, welches Kollokationen erkennt und beim Lernen hilft. Der L2-Lerner (Zweitsprache-Lerner) wird beim Schreiben von fremdsprachlichen Texten unterstützt. Durch die Möglichkeit, das KoKS-System als AddIn für Textverarbeitungsprogramme zu verwenden, kann es sogar direkt an Ort und Stelle bei der Textproduktion helfen.

### 11.2 Das KoKS-System und CALL

Der L2-Lerner erhält Hilfe bei der Übersetzung und beim Verständnis von Kollokationen in eine Fremdsprache. Durch das KoKS-System werden Kollokationskandidaten erkannt, welche dem Benutzer mit Übersetzungsbeispielen angezeigt werden. Dazu liefert es authentische Verwendungsbeispiele aus den parallelen Korpora.

Wir entschlossen uns, eine Demonstrationssoftware zu erarbeiten, welche das KoKS-System benutzt und im Anwendungsgebiet CALL angesiedelt ist. Das Szenario dafür sieht wie folgt aus:

Ein Nicht-Deutsch-Muttersprachler mit guten Englischkenntnissen (zum jetzigen Projektstand arbeitet das System mit Deutsch und Englisch), der Deutsch als Fremdsprache erlernt, versteht in einem Text einen bestimmten Satz nicht, d.h. er kann ihn nicht sinnvoll Wort für Wort mit einem Wörterbuch übersetzen. z.B.:

”Die Party fällt ins Wasser.” eine Wort-für-Wort-Übersetzung wäre: ”The party falls into the water.” Diese Übersetzung macht keinen Sinn. Hier kommt die KoKS-Demo-Applikation (siehe Abschnitt 10.1) ins Spiel, die dem Lerner hilft, den Satz richtig zu verstehen.

Wir denken, dass mit unserem Programm das Lernen von Sprachkonstruktionen im Kollokationsumfeld leichter gemacht wird.

### 11.3 Ausblick

Das KoKS-System kann vielseitig eingesetzt werden, z.B. als PlugIn für Textverarbeitungsprogramme oder als eine Art Point-To-Klick Anwendung (ähnlich dem Babylon-Translator). Das im Institut entwickelte LogoTax-System <sup>1</sup> benutzt das KoKS-System schon heute, um Verwendungsbeispiele aus den parallelen Korpora anzuzeigen. Die Ausgabe des KoKS-Systems ist anpassungsfähig, sodass alle Programme, die authentische Verwendungsbeispiele präsentieren wollen, grundsätzlich unterstützt werden können.

---

<sup>1</sup><http://www.cl-ki.uos/~LogoTax>



**Teil V**

**Dokumentation**



# Anhang A

## Systemadministration

### A.1 Software-Installation

Die Einrichtung der KOKS-Software auf einem Rechner ist sehr einfach, wenn man berücksichtigt, dass es sich um einen Prototypen handelt. Die meiste Arbeit besteht darin, eine vorhandene Linux-Installation so anzupassen, dass die Datenbank und die eingesetzten Interpreter aufgerufen werden können. Dazu sind im Folgenden hilfreiche Informationen zusammengetragen.

#### A.1.1 Überblick

- Programmiersprachen
- Systemdienste
- IMS-Tree-Tagger
- MySQL
- PostgreSQL
- KoKS-Software
- Konfiguration
- Funktionstest

#### A.1.2 Programmiersprachen

- Java 1.3
- Perl 5
- Python 1.5.2 bis 2.1.1
- Ruby 1.6
- Tcl 8.3

SuSE Linux 7.3 installiert standardmäßig Java 1.1.8. Java 1.3 wird aber im Paket IBMJava2-SDK mitgeliefert. Man muss aber noch

```
root@gorina:~> /usr/sbin/setDefaultJava IBMJava2
```

ausführen und die Variable `CREATE_JAVALINK` in der `/etc/rc.config` auf 'no' stellen.

**Zusätzliche Libraries****Java** JAXP (?)**Python** MySQLdb**Ruby** Ruby/Gtk, html-parser, OptionParser, XMLParser**Tcl** TclXML

MySQLdb muss eventuell von Hand installiert werden. Die Quellen gibt es z.B. auf <http://sourceforge.net/>. Paket entpacken und README lesen. Vermutlich muss die MySQL Bibliothek vorher installiert werden.

**A.1.3 Systemdienste**

Cron, CVS-Client (siehe auch Einstellungen für CVS), ...?

**A.1.4 IMS-Tree-Tagger**

Von der IMS-Seite herunterladen und unter `HOME/tagger` installieren. Die im KoKS-System verwendete Version benötigt die C-Bibliothek in der Version 5 (Paketname bei SuSE-Linux: `shlibs5`).

Wenn der Tagger in einem anderen Verzeichnis installiert ist, dann muss die Position in `align/DatabaseAPI/config.py` eingetragen und die Variable `TAGGERHOME` wie in A.1.7 beschrieben gesetzt werden.

**A.1.5 Datenbank****MySQL**

Den MySQL Datenbankserver gibt es in zwei Varianten. Wir benutzen die schnelle Variante, da wir keine Transaktionsunterstützung benötigen.

Wenn unter `/var` nicht genug Platz ist, dann sollte man einen SymLink `/var/mysql` oder `/var/lib/mysql` (je nach Version und Linux-Distribution verschieden) anlegen oder eine zusätzliche Partition mounten. Dabei Rechte, Owner und Gruppe beachten!

Bei SuSE-Linux muss in `/etc/rc.config` die Variable `START_MYSQL` auf `yes` gestellt werden. Danach `SuSEConfig` aufrufen. Mit `YaST2 – System – RC-Config Editor` wird dies automatisch erledigt. Dann entweder das ganze System neu starten oder manuell MySQL mit `/etc/init.d/mysql start` starten.

mysql-Kommandozeile starten:

```
$ mysql -u <user> -p <dbname>
```

Dabei wird immer ein Passwort abgefragt. Beim ersten Aufruf als einfach mit Enter weitergehen.

Passwort ändern als root in der Datenbank mysql:

```
update user set password = PASSWORD('pass') where user = 'root';
flush privileges;
```

Datenbank anlegen als root:

```
create db <dbname>
oder
create database <dbname>
```

User anlegen und ihnen Zugriff geben:

```
grant insert, select, update,
delete, alter, index, file, create, drop on <dbname>.* to
<user>@localhost identified by 'pass'; flush privileges;
```

Python - Treiber, ...?

Datenbank-Dump einspielen, z.B. als user koks in die db neu17b:

```
$ bzip2 -cd neu17b_wb.dump.bz2 | mysql -u koks -p neu17b
```

Wird die Datenbank intensiv benutzt, dann sollte vorher das Logging deutlich reduziert werden. Siehe dazu Abschnitt "The MySQL log files" im Handbuch. Für aktuelle Installationen ist auch der Unterabschnitt "The Binary Log" wichtig. Oder man schalten gleich das Logging ganz in /etc/my.cnf ab.

Achtung: die letzten Zeilen, die in mysql eingegeben wurden, werden in der Datei .mysql\_history gespeichert. Nach obiger Prozedur stehen dort also alle Paßwörter drin.

## PostgreSQL

Bei einigen SuSE-Versionen kann PostgreSQL nur bei der Erstinstallation eingespielt werden.

Bei SuSE-Linux kann das Datenverzeichnis in der rc.config eingestellt werden. Dort muss auch PostgreSQL zum Starten freigegeben werden. Danach SuSEConfig aufrufen.

Benutzer mit create... Tool als user postgres anlegen  
in pg\_options das Logging reduzieren

### A.1.6 KoKS-Software

Aus dem CVS müssen align, bin, db und share ausgecheckt werden. Sollen keine neuen Korpora aufgenommen oder Datenbanken angelegt werden, dann reichen align und bin aus. Um die Alignment-Tool benutzen zu können, müssen die Java-Quellen übersetzt werden:

```
koks@gorina:~> cd align/  
koks@gorina:~/align> make dds
```

Für die Demo-Applikation muss ein Webserver zur Verfügung stehen, auf dem Ruby-CGI-Skripte ausgeführt werden können. Eine statische HTML Seite, Java-Archive und die CGIs müssen eingerichtet werden. Eventuell muss im Java- und/oder Ruby-Code eingestellt werden, auf welchem Rechner der KoKS-Anfrageserver läuft.

### A.1.7 Konfiguration

In align/DatabaseAPI/config.py werden die meisten Pfade konfiguriert. Wenn der Tagger und die KOKS-Software direkt unter /home/koks eingerichtet wurden, dann kann einfach in einer vorhandenen Konfiguration der Hostname angepasst werden. Soll die Datei eventuell ins CVS eingechekkt werden, dann kopiert man natürlich die entsprechenden Zeilen. Die Konfiguration muss weiter angepasst werden, wenn die Datenbank gewechselt wird. Das Skript bin/delkorpus.py gibt eine Übersicht über die IDs der Wörterbücher und Korpora.

Damit auch Python-Skripte im Verzeichnis bin das Paket DatabaseAPI finden können, muss die Umgebungsvariable \$PYTHONPATH auf align gesetzt werden. Zusätzlich muss (warum?) bin aufgenommen werden, also z.B.

```
export PYTHONPATH=$HOME/align/:$HOME/bin/
```

Des Weiteren kann mit der Umgebungsvariablen \$TAGGERHOME angegeben werden, wo das Verzeichnis „tagger“ liegt. Default ist „/home/koks“, so dass der Pfad zu den IMS-Binaries „/home/koks/tagger/bin“ lautet.

### A.1.8 Funktionstest

In das Verzeichnis align wechseln und die Tagger-Anbindung testen:

```

koks@gorina:~/align> python
Python 2.1.1 (#1, Sep 24 2001, 05:28:47)
[GCC 2.95.3 20010315 (SuSE)] on linux2
Type "copyright", "credits" or "license" for more information.
>>> import sys, tagger
>>> t = tagger.tagline("Der Baum wächst schnell.", "de")
>>> t.writeTagzeilen(sys.stdout.write)
Der      ART      d
Baum     NN       Baum
wächst  VVFIN     wachsen
schnell ADJD     schnell
.        SATZ-P  .
<SATZ>
<segmentgrenze>
<ABSATZ>
>>>

```

In das Verzeichnis align wechseln und einen Query-Server starten:

```

koks@gorina:~/align> demo-qdb.py -s 0
Server auf Port 58597
*** Service started on Wed Nov  7 00:20:46 2001 ***

```

Typische Fehler: 1.) Modul MySQLdb fehlt. 2.) Konfiguration in align/DatabaseAPI/config.py stimmt nicht mit der eingerichteten Datenbank überein.

Auf zweiter Konsole den Zugriff auf die Datenbank testen:

```

koks@gorina:~/align> demo-qdb.py -c
Benutze Server ('localhost', 58597)
->t:segment;s:_ls;w:lemma=Fußballmannschaft
<2 items in list>
Fußballmannschaft
football team
->t:segment;s:text;w:phrases=Es gibt keine <i> Überraschungen </i> .,lang=de
<id:156179 type:wb>
Überraschungen NN      Überraschung
<segmentgrenze>
...

```

Je nach verwendeter Datenbank kann die Antwort des Servers abweichen. Fehler, insbesondere Python Exception, dürfen aber nicht auftreten.

## A.2 System-Administration

Da uns ein Projektrechner zur Verfügung stand, wurden allen Teilnehmern ein Benutzer-Account zur Verfügung gestellt. Für die Verwaltung von Sourcecode wurde ein CVS Repository<sup>1</sup> eingerichtet. Bei schwierigen Fragen empfehlen wir (Voss, 2001).

Protokolle der Projektsitzungen, Terminkalender und ähnliche Ressourcen sammelten wir in einem geschützten Bereich auf der Projektwebseite. Dort wurde auch ein CVS-Webzugang<sup>2</sup> integriert, um ein einfaches Navigieren und Betrachten der unterschiedlichen Code-Versionen zu ermöglichen.

<sup>1</sup><http://www.cvshome.org/>

<sup>2</sup><http://stud.fh-heilbronn.de/~zeller/cgi/cvsweb.cgi/>

## A.3 Linux Setup

Der Vollständigkeit halber hier die wichtigsten Einstellungen bei der Linux Installation von nunix:

```
*** nunix configuration Tue Aug 7 18:42:17 CEST 2001 ***
```

```
fdisk -l /dev/hda
=====
```

```
Disk /dev/hda: 255 heads, 63 sectors, 524 cylinders
Units = cylinders of 16065 * 512 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/hda1	*	1	13	104391	83	Linux
/dev/hda2		14	524	4104607+	5	Extended
/dev/hda5		14	79	530113+	82	Linux swap
/dev/hda6		80	462	3076416	83	Linux
/dev/hda7		463	524	497983+	83	Linux

```
fdisk -l /dev/sda
=====
```

```
Disk /dev/sda: 255 heads, 63 sectors, 527 cylinders
Units = cylinders of 16065 * 512 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/sda1		1	17	136521	82	Linux swap
/dev/sda2		18	213	1574370	fd	Linux raid autodetect
/dev/sda3		214	246	265072+	83	Linux
/dev/sda4		247	527	2257132+	5	Extended
/dev/sda5		247	331	682731	83	Linux
/dev/sda6		332	527	1574338+	fd	Linux raid autodetect

```
cat /etc/fstab
=====
```

/dev/sda1	swap		swap	defaults	0	0
/dev/hda5	swap		swap	defaults	0	0
/dev/sda3	/		ext2	defaults	1	1
/dev/sda5	/var		ext2	defaults	1	2
/dev/md1	/usr		ext2	defaults	1	2
/dev/hda1	/boot		ext2	defaults	1	2
/dev/hda6	/home		ext2	defaults	1	2
/dev/hda7	/mnt/tmp/		ext2	defaults	1	2
/dev/scd0	/cdrom		auto	ro,noauto,user,exec		
0	0					
/dev/fd0	/floppy		auto	noauto,user	0	0
none	/proc		proc	defaults	0	0
# End of YaST-generated fstab lines						

```
# KOKS Datenbank
/dev/md0          /data                ext2                defaults 1 2

# KOKS Daten (Korpora, Backups etc.)
udeis:/export    /mnt/udeis            nfs
defaults,nodev,nosuid,soft
```

```
cat /etc/raidtab
=====
```

```
raiddev /dev/md0
    raid-level      0
    nr-raid-disks   2
    persistent-superblock 1
    chunk-size      4
    device           /dev/sdb1
    raid-disk        0
    device           /dev/sdc1
    raid-disk        1
```

```
raiddev /dev/md1
    raid-level      linear
    nr-raid-disks   2
    chunk-size      4
    persistent-superblock 1
    device           /dev/sda2
    raid-disk        0
    device           /dev/sda6
    raid-disk        1
```

```
cat /etc/lilo.conf
=====
```

```
# LILO Konfigurations-Datei
# Start LILO global Section
#append="auto mem=256M ether=10,0xe000,eth0"
#append="mem=256MB sb=220,5,1,1"
append="mem=256MB root=/dev/sda3 sb=220,5,1,1"
boot=/dev/hda
#compact          # faster, but won't work on all systems.
linear
read-only
prompt
timeout=50
vga = normal      # force sane state
# End LILO global section

# Linux bootable partition config begins
image = /boot/vmlinuz-2.2.18
root = /dev/sda3
label = linux2218
# Linux bootable partition config ends

# Linux bootable partition config begins
```

```
image = /boot/vmlinuz-2.2.16
root = /dev/sda3
label = linux2216
# Linux bootable partition config ends

## Linux bootable partition config begins
#image = /boot/vmlinuz-2.4.0
#root = /dev/sda3
#label = linux240
## Linux bootable partition config ends

crontab -l
=====
# check free disk space - Joachim
27 04 * * * /root/bin/checkdf.sh
# run logrotate to keep log files in /var/log small - Arno
# run every night at 4
00 04 * * * /usr/local/sbin/logrotate /usr/local/etc/logrotate.conf
```

# Anhang B

## Meta-Dokumentation

### B.1 Dokumentation

Die vorliegende Dokumentation wurde mit  $\text{\LaTeX}$  erstellt. Nachdem die Projektmitglieder sich auf eine Gliederung geeinigt hatten, wurde eine entsprechende Verzeichnisstruktur im CVS angelegt. Die Hauptdatei der Dokumentation bindet alle weiteren Dateien aus den Verzeichnissen ein. Ein Makefile regelt die Übersetzung der Hauptdatei in PostScript.

Durch den Einsatz von CVS wurde die gemeinsame Arbeit an der Dokumentation erheblich erleichtert. Es wurden eigene Makros definiert, die es ermöglichen, Autorennamen und Fertigstellungsgrad am Seitenrand zu vermerken, so dass deutlich wird, wer eigentlich was geschrieben hat. Weitere Makros dienen dazu, die Schreibung von KoKS zu vereinheitlichen, Fragwürdige Stellen zu markieren und wichtige Termini hervorzuheben.

#### B.1.1 Makrodefinitionen

Die Makros sind in der Hauptdatei definiert:

```
% einige neue Befehle
\newcommand{\by}[1]{\marginpar{[#1]}}           % Autor kennzeichnen
\newcommand{\buzzword}[1]{\textit{#1}}         % wichtiges Wort
\newcommand{\programm}[1]{\texttt{#1}}         % Programmname
\newcommand{\hmm}{\marginpar{\scalebox{.085}{\includegraphics{review-me}}}}
                                                    % überarbeitungsbedürftige
                                                    % Textstellen kennzeichnen
\newcommand{\koks}{KoKS\xspace}                % KoKS-Makro zur Vereinheit-
                                                    % lichung der Schreibweise
\newcommand{\todo}[1]{\textbf{TODO:}\xspace[1]}
\newcommand{\status}[1]{\marginpar{fertiggestellt zu #1 \%}}
                                                    % Fertigstellungs-Prozent-
                                                    % angebe am Textrand
```

### B.2 Präsentationen

#### B.2.1 Präsentation am 13.12.2000

Die erste Präsentation stellte die Pläne und Ideen unseres Projektes vor. Zum Erstellen der Folien benutzen wir Powerpoint und Excel<sup>1</sup>. Dabei stellte es sich heraus, dass man zwar schnell eine Grundversion hat, für die Details musste aber relativ viel Zeit aufgewandt werden. Außerdem konnten so nur zwei Teilnehmer daran arbeiten.

---

<sup>1</sup><http://www.microsoft.com/>

### **B.2.2 Präsentation am 24.06.2001**

Für den Hochschul-Informationstag (HIT) wurde eine Präsentation erstellt, die sowohl potentielle Studienanfänger wie auch die interessierte Öffentlichkeit ansprechen sollte. Aufgrund der Erfahrungen bei der ersten Präsentation, entschieden wir uns, diesmal HTML-Seiten zu erstellen. Dafür wurde das Programm Netobjects Fusion<sup>2</sup> eingesetzt. Damit konnten die Seiten schnell generiert werden, allerdings sind manuelle Änderungen nur schwer möglich.

### **B.2.3 Präsentation am 29.08.2001**

Im Rahmen des Pre-Conference Workshops der „Special Interest Group Language Processing“ (SigLP) zur EuroCALL 2001 stellte das Projekt seine Ergebnisse vor. Diesmal wurde die Präsentation mit L<sup>A</sup>T<sub>E</sub>X und dem Prosper-Paket<sup>3</sup> gebaut. Rückblicked erscheint diese Lösung empfehlenswert.

---

<sup>2</sup><http://www.netobjects.com/>

<sup>3</sup><http://prosper.sourceforge.net/>

## Anhang C

# Korpora – Dokumentation

### C.1 Tools und Optionen

#### C.1.1 Korpus-Verwaltung

Die Skripte für das automatische Erstellen der `index.xml` Dateien sind sehr abhängig vom Aufbau der jeweiligen Teilkorpora. Daher ist eine allgemeine Beschreibung schlecht möglich. Es sei auf Abschnitt I.1 im Code-Anhang als Beispiel verwiesen. Außerdem gibt Abschnitt 3.4.1 einen Überblick über das prinzipielle Vorgehen.

#### C.1.2 Korpus-Auswertung

Im Folgenden werden einfache Methoden vorgestellt, wie sich die quantitativen Daten zu den Korpora produzieren lassen. Grundlage ist meistens die Indexdatei `index.xml`.

Um die Größen aller Rohdaten zu ermitteln, sammelt man einfache alle Dateinamen auf und addiert die Dateigrößen:

```
koks@nunix:~/korpus/mci > idx-ls.tcl -t | xargs ls -s | awk -f ../count.awk
```

Das notwendige `awk`-Skript ist schnell zusammengebaut, natürlich lassen sich diese Aufgaben auch mit anderen Skriptsprachen einfach realisieren.

```
BEGIN { s = 0 }  
{ s += $1 }  
END { s foo }
```

Um die Anzahl der Zeilen, Wörter und Zeichen zu zählen, kann man einfach das Programm `wc` benutzen:

```
koks@nunix:~/korpus/mci > idx-ls.tcl -t | xargs wc
```

Des Weiteren lassen sich die getaggten Daten: Um bspw. die Zahl der Tokens zu ermitteln, werden alle Zeilen mit drei Spalten gezählt:

```
koks@nunix:~/korpus/mci > idx-ls.tcl -l de | cut -d ':' -f 2 | \  
sed 's/eci/tag/' | xargs cat | awk 'BEGIN{c=0} NF=3{c+=1} END{print c}'
```

Ebenso kann die Anzahl der Segmente gezählt werden:

```
koks@nunix:~/korpus/mci > idx-ls.tcl -l de | cut -d ':' -f 2 | \  
sed 's/eci/tag/' | xargs grep "<segmentgrenze>" | wc -l
```

Ähnliche Auswertungen könnte man mit den alignten Dateien durchführen.

### C.1.3 XML Tools

Wie bereits in Abschnitt 3.6.1 beschrieben, werden die Informationen über die Korpora und die darin enthaltenen Dateien in XML-Dateien abgelegt. Dieser Abschnitt beschreibt die Programme, mit denen solche XML-Dateien erzeugt und angezeigt werden können. Abschnitt C.2.1 beschreibt die zu Grunde liegenden DTDs.

#### idx-ls.tcl

idx-ls.tcl zeigt den Inhalt einer index.xml Datei. Somit kann man sich einen Überblick verschaffen, was für Dateien in einem (Teil-)Korpus vorhanden sind.

Verwendung:

```
koks@nunix:~ > idx-ls.tcl -h
Usage: idx-ls.tcl [-c] [-f] [-s] [-t] [-l <lang>] [index-file]
-c: compute checksum and only list changed files
-f: print out file format of each file
-l <lang>: only list entries of language <lang> (de, en, bi)
-s: print some statistics in the end
-t: terse mode, only list file names
```

Beispiel: Zeige alle deutschen Dokumente im MCI-Korpus und ihre Formate:

```
koks@nunix:~/korpus/mci > idx-ls.tcl -f -l de -s
de: (sgml) mul01/mul01a03.eci

de: (sgml) mul01/mul01a06.eci

de: (sgml) mul01/mul01b07.eci

de: (sgml) mul01/mul01b38.eci

de: (sgml) mul06/mul06e.eci

de: (sgml) mul09/mul09e.eci
--> 6 documents (6 parts)
```

#### idx-add.tcl

Das Tools idx-add.tcl dient zum (halb-)automatischen Erstellen von index.xml Dateien. Als Parameter kann man Angaben zu den Korpus-Dateien (Autor, Quelle, etc.) machen, die dann in der index.xml Datei entsprechend eingefügt werden.

**Verwendung** idx-add.tcl bietet eine Reihe von Optionen:

```
koks@nunix:~ > idx-add.tcl -h
Usage:
idx-add.tcl [-x] [info-options] [-u <use>] -p -l <lang> <file1> <format1> \
<file2> <format2> ...

General Options:
-x: short mode, skip document header and footer
-p: start a new part
-m: generate id (checksum) using md5
-l: set language of part
-u: mark document for use
```

```

Info Options:
-a: set author
-c: set comment
-d: set date
-s: set source
-t: set source type

<source_type> may be one of "url cdrom unknown"
<format> may be one of "plain html sgml pdf ps doc unknown"
<lang> may be one of "de en bi"
<use> may be one of "yes no"

```

**Beispiel** Erstelle eine `index.xml` Datei. Das Dokument besteht aus einem englischen und einem deutschen Teil. Die deutsche Version liegt in der Datei `siddharta_de.txt` im ASCII Format. Die englische Version liegt ebenfalls im ASCII Format in der Datei `siddharta_en.txt`. Autor des Dokuments ist „Hermann Hesse“, es trägt als Datum 1922 und stammt vom Flohmarkt.

```

koks@nunix:/tmp > idx-add.tcl -a "Hesse" -c "vom Autor signiert" -d 1922 \
-s Flohmarkt -t unknown -p -l de siddharta_de.txt plain \
-p -l en siddharta_en.txt plain

```

liefert

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE mapping SYSTEM "/home/koks/share/mapping.dtd">
<mapping>
<document>
<part lang="de">
<file>
<filename format="plain">siddharta_de.txt</filename>
</file>
</part>
<part lang="en">
<file>
<filename format="plain">siddharta_en.txt</filename>
</file>
</part>
<info>
<author>Hesse</author>
<comment>vom Autor signiert</comment>
<date>1922</date>
<source type="unknown">Flohmarkt</source>
</info>
</document>
</mapping>

```

Besteht eine sprachliche Variante aus mehreren Teilen, so gibt mit einfach alle Dateien der Reihe nach mit ihrem Format an:

```

koks@nunix:/tmp > idx-add.tcl -c Test -p -l de de/part1 plain de/part-2 ps \
de/part3 html -p -l en en/part1 pdf

```

ergibt

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE mapping SYSTEM "/home/koks/share/mapping.dtd">
<mapping>
<document>
<part lang="de">
<file>
<filename format="plain">de/part1</filename>
</file>
<file>
<filename format="ps">de/part-2</filename>
</file>
<file>
<filename format="html">de/part3</filename>
</file>
</part>
<part lang="en">
<file>
<filename format="pdf">en/part1</filename>
</file>
</part>
<info>
...
</info>
</document>
</mapping>

```

Der Schalter `-m` erstellt automatisch Prüfsummen. Das ist praktisch, um festzustellen, ob sich eine Datei geändert hat:

```
koks@nunix:/tmp > idx-add.tcl -m -p -l de d.txt plain -p -l en e.txt plain
```

liefert

```

<!DOCTYPE mapping SYSTEM "/home/koks/share/mapping.dtd">
<mapping>
<document>
<part lang="de">
<file>
<filename format="plain">d.txt</filename>
<checksum method="md5">83e079c6a4aa895e79195effe2c7c6b1</checksum>
</file>
</part>
<part lang="en">
<file>
<filename format="plain">e.txt</filename>
<checksum method="md5">78c6e6b2616628b1891cf0c303d0d8d0</checksum>
</file>
</part>
<info>
...
</info>
</document>
</mapping>

```

Die Prüfsummen werden von `idx-ls.tcl` bei Bedarf ausgewertet; hat sich bspw. die englische Fassung geändert, sieht man:

```
koks@nunix:/tmp > idx-ls.tcl
de: d.txt
en: e.txt
koks@nunix:/tmp > idx-ls.tcl -c
en: e.txt
```

### **idx-repair.tcl**

Liest eine halbfertige `index.xml` Datei ein und fügt den korrekten Vorspann hinzu. Das ist nötig, wenn vorher mehrere mit `idx-add.tcl -x` erstellte Dateien zusammengefügt werden.

### **meta-ls.rb**

Die Datei `metaindex.xml` enthält Verweise auf alle im Korpus-Baum vorhandenen `index.xml` Dateien. Mit `meta-ls.rb` kann man sich den Inhalt von `metaindex.xml` anzeigen lassen:

Beispiel:

```
koks@nunix:~/korpus > meta-ls.rb
de-news
    de-news/1996/index.xml
    de-news/1997/index.xml
    de-news/1998/index.xml
    de-news/1999/index.xml
    de-news/2000/index.xml
eu
    eu/90/index.xml
    eu/91/index.xml
    eu/92/index.xml
    eu/93/index.xml
    eu/94/index.xml
    eu/95/index.xml
    eu/96/index.xml
    eu/97/index.xml
    eu/98/index.xml
    eu/99/index.xml
    eu/00/index.xml
    eu/01/index.xml
nato
    nato/1996/index.xml
    nato/1997/index.xml
    nato/1998/index.xml
    nato/1999/index.xml
mci
    mci/index.xml
wb
    wb/index.xml
bible
    bible/index.xml
lql
    lql/index.xml
verbmobil
    verbmobil/index.xml
```

### C.1.4 Sprachklassifikation

Diese Tools dienen dazu, die Sprache eines Textdokumentes festzustellen. Zuerst erfolgt eine Trainingsphase mit je einem Dokument pro Sprache. Dann können unbekannte Dokumente analysiert und klassifiziert werden.

#### ngram.rb

ngram.rb erstellt aus dem Textdokument eine Liste von n-Grammen mit ihren jeweiligen Häufigkeiten. Das wird für die Trainings- und die Testdokumente benötigt, wobei zum Testen eine geringe Datenmenge ausreicht.

Anwendung:

```
koks@nunix:~/artus > ngram.rb -h
Usage: ngram.rb [options]
  -i, --input=FILENAME      input file name
  -o, --output=FILENAME     output file name
  -b, --bytes=VALUE         read at most VALUE bytes
  -l, --len=VALUE           build n-grams of VALUE length
  -n, --normalize            normalize n-grams
  -h, --help                show this message
```

Mit dem Schalter `-b` kann man die einzulesende Datenmenge beschränken. In der Praxis haben sich ca. 500 Bytes als ausreichend herausgestellt. Der Schalter `-l` stellt die Länge der zu erstellenden n-Gramme (Unigramm, Bigramm etc.) ein. Wir haben die Klassifikation mit Trigrammen durchgeführt. Schließlich können die n-Gramme mit dem Schalter `-n` normalisiert werden. Normalisieren bedeutet hier:

- alles in Kleinbuchstaben wandeln
- Umlaute in '#' wandeln
- allen Whitespace gleich behandeln

Desweiteren werden alle Nichtbuchstaben (Sonderzeichen, Zahlen, etc.) ignoriert.

#### cfile.rb

Hat man mit ngram.rb sowohl n-Gramm Listen für die Trainings- als auch die Testdokumente erstellt, kann man mit cfile.rb die Klassifikation starten.

Anwendung:

```
koks@nunix:~/artus > cfile.rb -h
Usage: cfile.rb [options]
  -d, --data=FILENAME       data file names [default: 'datafiles']
  -t, --test=FILENAME       test file name [default: 'testfiles']
  -n, --nosmooth            do not smooth training data
  -v, --verbose              be verbose
  -h, --help                show this message
```

Der Schalter `-n` führt dazu, dass die Menge aller gefundenen n-Gramme nicht geglättet wird. Glätten der Trainingsdaten ist empfehlenswert, siehe dazu auch Abschnitt 3.5.1

Beispiel:

```
koks@nunix:~/artus > cfile.rb -d datafiles -t testfiles -v
Reading data file 'data/data.de'...(2524, 43594)
Reading data file 'data/data.en'...(2186, 37861)
Smoothing training data...
Normalizing training data...
```

```

Reading test file '/mnt/tmp/artus-test/p96-165d.dat'...
Cleaning test data...
Normalizing test data...
Calculating distances...
dist[0] = 1.606591339
dist[1] = 3.301345184
/mnt/tmp/artus-test/p96-165d.dat: de
Reading test file '/mnt/tmp/artus-test/p96-165e.dat'...
Cleaning test data...
Normalizing test data...
Calculating distances...
dist[0] = 3.564639446
dist[1] = 1.58033287
/mnt/tmp/artus-test/p96-165e.dat: en

```

**classifier.rb**

`classifier.rb` wird von `cfile.rb` benutzt und stellt die Methoden zur Verfügung, um die n-Gramm Daten zu klassifizieren. Auch die Client-Server Version benutzt es. Es kann nicht direkt aufgerufen werden.

**cserver.rb**

`cserver.rb` ist eine Server-Variante von `cfile.rb`. Die Idee ist, einen Klassifikations-Server laufen zu lassen, der über einen Socket zu klassifizierende Daten (= n-Gramm Listen) erhält und ein Ergebnis zurück liefert. Der Vorteil gegenüber der stand-alone Version ist, dass die teilweise großen Mengen an Trainingsdaten nur einmal beim Starten eingelesen werden müssen und nicht bei jeder zu bearbeitenden Test-Datei.

Anwendung:

```

koks@nunix:~/artus > cserver.rb -h
Usage: cserver.rb [options]
  -d, --data=FILENAME      data file names [default: 'datafiles']
  -p, --port=PORT          listening port [default: 9999]
  -n, --nosmooth           do not smooth training data
  -v, --verbose            be verbose
  -h, --help              show this message

```

Die Schalter `-d`, `-n`, `-v` haben die gleiche Bedeutung wie beim Programm `cfile.rb`. Der Schalter `-p` legt den Port fest, auf dem der Server läuft.

**cclient.rb**

Das zugehörige Client-Programm ist `cclient.rb`. Es liest die Daten aus der angegebenen Test-Datei und sendet sie an den Server. Von dort erhält es ein Ergebnis und gibt es aus.

Anwendung:

```

koks@nunix:~/artus > cclient.rb -h
Usage: cclient.rb [options]
  -t, --test=FILENAME      test file names [default: 'testfile']
  -s, --server=ADDRESS     server address
  -p, --port=PORT          server port [default: 9999]
  -h, --help              show this message

```

Der Schalter `-t` gibt den Namen der Test-Datei an. Mit den Schaltern `-s` und `-p` legt man die Adresse fest, unter der der Servers erreichbar ist.

Zur Sprachidentifikation gibt es also zwei Möglichkeiten: Alle Dateien einzeln mit `cfile.rb` bearbeiten, oder einen Server (`cserver.rb`) starten und die Daten mit `cclient.rb` einzeln dem Server zur

Auswertung schicken. Das Client-Server Paar eignet sich so auch zur einfachen Anbindung an eine Weboberfläche, z.B. per CGI.

## C.2 Dateiformate

### C.2.1 DTD

Document Type Definitions (DTDs) legen den Aufbau einer XML-Datei fest. Erfüllt ein XML-Dokument alle in einer DTD spezifizierten Anforderungen, so nennt man dieses Dokument *valid*. Für die Korpus-Verwaltung wurden zwei DTDs erstellt, die im Folgenden vorgestellt werden. Die in Abschnitt C.1.3 beschriebenen Programme basieren auf diesen DTDs.

#### mapping.dtd

Die erlaubte Struktur von `index.xml` wird in der DTD `mapping.dtd` beschrieben. Die DTD spezifiziert dabei Entities, Elemente und Attribute. Die Entity-Definitionen legen die erlaubten Werte für verschiedene Attribute wie Sprache, Format, etc. fest:

```
<!-- some information remains unknown -->
<!ENTITY % unknown.att "unknown">

<!-- flag indicating whether a document can be used for further processing -->
<!ENTITY % use.att "yes|no">

<!-- document kind is either lexicon, phrase, or text -->
<!ENTITY % kind.att "lexicon|phrase|text">

<!-- documents can be either German (de) or English (en)
      or both languages in one file (bi) -->
<!ENTITY % lang.att "de|en|bi">

<!-- each part can use its own tag set -->
<!ENTITY % set.att "ims|lq1">

<!-- allowed file formats, list should be extended as needed -->
<!ENTITY % format.att "plain|html|sgml|pdf|ps|doc|tagged|%unknown.att;">

<!-- recognized encodings for text files -->
<!ENTITY % enc.att "iso-8859-1|%unknown.att;">

<!-- allowed types for document source -->
<!ENTITY % type.att "url|cdrom|%unknown.att;">

<!-- allowed methods for document checksums -->
<!ENTITY % method.att "cksum|md5|sha1">
```

Die Element-Definitionen bestimmen den Aufbau einer `index.xml` Datei. Das Wurzel-Element ist `<mapping>`, weil die prinzipielle Aufgabe der `index.xml` Datei die Zuordnung von englischen Dokumenten (bzw. Teilen) zu ihren korrespondierenden deutschen Teilen ist. Die DTD ist allerdings deutlich flexibler: Eine Zuordnung (`mapping`) kann aus beliebig vielen Teilen bestehen, in unserem Fall beschränkten wir uns jedoch auf zwei.

```
<!ELEMENT mapping (document+)>
```

Bestandteile eines Mappings sind Dokumente. Ein Dokument ist dabei eine abstrakte Einheit, sozusagen der logische Zusammenhang von verschiedenen Teilen. Ein solches logisches Dokument ist also nicht mit einer tatsächlichen Datei gleichzusetzen. Zu jedem Dokument können optionale Angaben gemacht werden: Eine eindeutige ID (wird vom Korpus-Browser verwendet), ein Flag, ob das Dokument benutzt werden soll und eine generelle Unterscheidung über den Inhalt des Dokuments. Letztere dient dazu, Lexika von richtigen Texten zu unterscheiden.

```
<!-- each document consists of one or more parts and additional info -->
<!ELEMENT document (part+,info?)>
<!ATTLIST document
    id ID #IMPLIED
    use (%use.att;) #IMPLIED
    kind (%kind.att;) #IMPLIED>
```

Jeder Teil eines Dokuments besteht aus einer oder mehreren Dateien. Hierin spiegelt sich die Aufteilung eines abstrakten Dokuments als Ganzes in tatsächlich vorhandene Dateien wieder. Jeder Teil muss hinsichtlich seiner Sprache spezifiziert werden, optional kann man angeben, welches Tagset benutzt werden soll. Das tagset Attribut wird benötigt, weil die Wörterbücher teilweise schon eigene Tagsets mitbringen und daher nicht mehr mit dem Tagger bearbeitet werden müssen.

```
<!-- each part consists of one or more files in a specified language -->
<!ELEMENT part (file+)>
<!ATTLIST part
    lang (%lang.att;) #REQUIRED
    tagset (%set.att;) #IMPLIED>
```

Im KoKS-Projekt benutzen wir format="tagged" zusammen mit tagset="lql" für die LQL-Wörterbücher. So weiß das Normalisierungs- Programm |preproc.py—, dass diese Dateien nicht mehr getaggt werden müssen. Die unterschiedlichen Tagsets werden auch beim Eintragen in die Datenbank bzw. beim Abfragen berücksichtigt.

Die Unterelemente eines Teils <part> sind Dateien <file>. Man könnte an dieser Stelle bereits den Dateinamen angeben, es wurde aber eine weitere Unterteilung in ein <filename> Tag und einen optionalen Block für Prüfsummen vorgenommen. Auf diese Art von Entscheidung Tag vs. Attribut trifft man immer wieder beim Erstellen von DTDs.

```
<!-- a file is identified by its name and an optional checksum -->
<!ELEMENT file (filename,checksum?)>
```

Das <filename> Tag schließlich enthält den Dateinamen. Eine Angabe zum Dateiformat (ps, pdf, html, ...) ist zwingend erforderlich. Zusätzlich kann das Encoding spezifiziert werden, was hilfreich bei der Behandlung von Umlauten ist.

```
<!-- filename contains the relative path to the described file,
    also there is an optional encoding tag -->
<!ELEMENT filename (#PCDATA)>
<!ATTLIST filename
    format (%format.att;) #REQUIRED
    enc (%enc.att;) #IMPLIED>
```

Das <checksum> Tag dient dazu, Dokumente eindeutig zu identifizieren. Im Prinzip ist es eine ID für die einzelnen Dateien, allerdings optional. Durch Vergleich der hinterlegten Prüfsumme (MD5, SHA) mit dem aktuellen Wert kann man feststellen, ob das Dokument verändert wurde, bspw. durch manuelle Nachbearbeitung. Das Programm idx-add.tcl kann das <checksum> automatisch mit einer berechneten Prüfsumme eintragen. Im Gegenzug wertet idx-add.tcl bei Aufruf mit dem Schalter -c ein vorhandenes <checksum> Tag aus und listet nur Dateien auf, deren Prüfsumme sich geändert hat.

```
<!-- checksum can be used to uniquely identify documents -->
<!ELEMENT checksum (#PCDATA)>
<!ATTLIST checksum method (%method.att;) #REQUIRED>
```

Der Info Block zu jedem Dokument wird genutzt, um Informationen über Autor, Herkunft, Datum und weitere Kommentare abzulegen. Herkunftsangaben sind wichtig, um Dokumente bei Verlust wiederzubeschaffen, aber auch für juristische Fragen (Copyright).

```
<!-- the info tag can be further refined, e.g. for source information -->
<!ELEMENT info (author,comment,date,source)>

<!ELEMENT author (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT date (#PCDATA)>

<!-- type of source can remain unspecified -->
<!ELEMENT source (#PCDATA)>
<!ATTLIST source type (%type.att;) #IMPLIED>
```

### meta.dtd

Die DTD für den Meta-Index ist einfacher: Sie besteht nur aus den Elementen `korpus` und `file` sowie dem Wurzel-Element `meta-index`. Der Meta-Index enthält Verweise auf alle `index.xml` Dateien im gesamten Korpus-Baum.

```
<!ELEMENT meta-index (korpus+)>

<!ELEMENT korpus (file*)>
<!ATTLIST korpus
    name CDATA #REQUIRED>

<!ELEMENT file (#PCDATA)>
```

## C.2.2 Sprachklassifikation

Um das Zusammenspiel der Programme und Dateien bei der Sprachidentifikation zu erläutern, hier ein Beispiel. Es sollen die Dateien `p96-165d.dat` und `p96-165e.dat` getestet werden. Die Trainingsdaten liegen in den Dateien `data/data.de` und `data/data.en` vor.

`cfile.rb` liest zwei Dateien ein: Eine Datei enthält eine Liste mit den Namen der Testdateien (je ein Dateiname mit Pfad pro Zeile). Die andere Datei enthält ein Kürzel, welches die Sprache angibt, und den zugehörigen Dateinamen (durch Tab getrennt).

Beispiel:

```
koks@nunix:~/artus > cat testfiles
/mnt/tmp/artus-test/p96-165d.dat
/mnt/tmp/artus-test/p96-165e.dat
koks@nunix:~/artus > cat datafiles
de    data/data.de
en    data/data.en
```

Der Inhalt von `testfiles` ist also eine Liste von Dateinamen. Die dort aufgelisteten Dateien enthalten Testdaten. Die Datei `datafiles` enthält eine Liste von ID und Dateinamen Pärchen. Diese Dateien sind Trainingsdaten – meistens deutlich größer als Testdaten. Die ID ist ein String, der ausgegeben wird, wenn die Testdaten diesen Trainingsdaten am ähnlichsten sind.

Sowohl Test-Dateien als auch Trainings-Dateien enthalten n-Gramm Listen und werden von `ngram.rb` automatisch erstellt. Sie enthalten pro Zeile ein n-Gramm und (durch Doppelpunkt getrennt) die jeweilige Häufigkeit:

```
koks@nunix:~/artus > cat data/data.de
...
dab:4
daf:3
dah:1
dal:2
dam:3
dan:1
dar:15
das:52
dat:5
dau:4
dav:1
daz:4
de :16
dea:3
def:3
dei:1
...
```

# Anhang D

## Normalisierung – Dokumentation

### D.1 Tools und Optionen

#### D.1.1 normalize.py

Dieses Skript wandelt eine ihm übergebene Datei aus dem angegebenen Format in das KoKS-Dateiformat um. Es entstehen 2 Dateien: <datei>.asc, die den konvertierten Text enthält, und <datei>.asc2, die zur Weiterverarbeitung benutzt wird und auch das <DOKENDE>-Markup besitzt.

Beispielaufruf: `normalize.py koks.html html`

#### D.1.2 preproc.py

`preproc.py` erwartet als Parameter beliebig viele XML-Dateien, die dann der Reihe nach geparkt werden. Die dort angegebenen Dateien werden mithilfe von `normalize.py` nach ASCII umformat. Dann werden die Dateien getaggt, und alignt. Die getaggten Dateien heissen <datei>.tag, die alignten Dateien heissen <datei>.tag.al. Schliesslich werden die Dokumente in die Datenbank eingetragen. Damit dies nicht doppelt passiert, werden leere Dateien <datei>.tag.db angelegt.

Wenn es eine Datei `/tmp/preproc_dbname.running` gibt, läuft das Skript entweder noch, oder es wurde nicht ordnungsgemäß beendet.

Achtung: Der Tagger muss installiert und in `config.py` eingetragen sein.

```
Usage: preproc.py [-a] [-c|-n] [-i <id>] [-k <kind>] <xml-files> ...
-h          zeigt die Hilfe
-a          führt kein(!) Alignment aus
-c          führt nur die Konvertierung ins KOKS-Format durch, d.h.
           Tagger, Aligner und Enter2Db werden asugelassen
-n          führt alle Schritte einschliesslich Alignen aus, trägt
           aber nichts in die DB
-i <id>    ändert den Namen, der run-Datei (/tmp/preproc_<id>.running)
           sollte nur verwendet werden, wenn nichts in die DB
           geschrieben wird, d.h. zusammen mit -n
-k <kind>  dient dazu, Dokumente nach ihrer Art (phrase, lexicon
           text) auszuwählen
```

#### D.1.3 korpus2db.py

Das Skript trägt zwei alignte Korpora in die Datenbank ein. Es orientiert sich an den Markierungen <segmentgrenze> und <SATZ>, um die Segment- bzw. Satzgrenzen zu finden. Auch Markup innerhalb von <KOKS></KOKS> wird ignoriert.

Übergibt man dem Skript den Parameter `-e` und 2 Dateien, fragt es zunächst nach Angaben zu den Dateien und trägt sie dann als alignten Korpus in die Datenbank ein.

Beispielaufruf: `korpus2db.py de.tag.al en.tag.al`

#### D.1.4 `py2pl.py`

Liest die Konfiguration aus `config.py` ein und schreibt sie in Perl-Format nach `stdout`. So kann ein Perl-Skript mit

```
#!/usr/bin/perl -w
my $config = require("config.pl");
```

die selben Konfigurationswerte verwenden, Bsp:

```
my $user=$config->{dbuser};
```

Das generierte Skript `config.pl` überprüft selbst, ob es älter als die zu Grunde liegende Konfigurations-Datei `config.py` ist. In diesem Fall wird eine Warnung ausgegeben, so dass der Benutzer sieht, dass die verwendete (Perl-)Konfiguration möglicherweise nicht mehr der (Python-)Konfiguration entspricht. Die Position von `config.py` kann mit einer Umgebungsvariable bestimmt werden. Die Werte aus `config.py` werden als Perl-Hash gespeichert.

`config.pl` wird z.B. von `umlaut.pl` (s. Kap. E.1.3) benutzt, um die richtigen Angaben für die Datenbankverbindung zu bentuzen.

# Anhang E

## Tagging – Dokumentation

### E.1 Die Tools im Einzelnen

Der Prozess des Taggens gliedert sich in einzelne Teilprozesse, die von jeweils eigenen Tools übernommen werden. Sie sind zusammengefasst in den Shell-Skripten `bin/tree-tagger-german` und `bin/tree-tagger-english`. Die Teilprozesse sind im einzelnen:

- Tokenisierung
- Tagging
- Lemmatisierung
- Satzendenerkennung

In den Ablauf dieser Prozesse eingebunden ist eine Sonderzeichenrekonstruktion.

Die Tokenisierung übernimmt das Tool `tagger/bin/separator-punctuation`; das Ergebnis wird von dem Tool `bin/metatag-correction` korrigiert. Es folgt die Sonderzeichenrekonstruktion mit dem Tool `bin/umlaut.pl`, anschließend das Taggen und Lemmatisieren mit dem Tool `tagger/bin/tree-tagger`. Die Satzendenerkennung übernimmt das Tool `bin/punkt-tagger-german` bzw. `bin/punkt-tagger-english`.

Der Tokenisierer und der Tagger sind Programme des IMS Stuttgart; daher befinden sie sich im Verzeichnis `tagger/bin`. Die Korrektur der Tokenisierung, die Sonderzeichenrekonstruktion und die Satzendenerkennung sind eigene Tools.

#### E.1.1 Tokenisierung: `tagger/bin/separator-punctuation`

Die Aufgabe der Tokenisierung besteht darin, einen zu taggenden Text in einzelne Token zu segmentieren. Token sind Wörter und Interpunktionszeichen. Was ein Wort ist, ist nicht immer eindeutig zu entscheiden: Problematisch sind beispielsweise diskontinuierliche Verben („Er schloss die Tür ab“) oder durch Bindestriche zusammengesetzte Wörter („Drei-Tage-Bart“). Schwierigkeiten treten auch bei der Isolierung der Interpunktionszeichen auf: Punkte können das Satzende markieren, ebenso aber z.B. Ordinalzahlen oder Abkürzungen.

Der von uns verwendete Tokenisierer stammt vom IMS in Stuttgart. Er erkennt Abkürzungen mit Hilfe einer Wortliste von Abkürzungen. Solche Listen stellt das IMS für Englisch und Deutsch als plain text zur Verfügung; sie können also leicht verändert werden. Die für das Deutsche enthält 1301 Einträge, die für das Englische 118. Partikel diskontinuierlicher Verben werden als eigenes Wort angesehen und entsprechend segmentiert. Die Kommandosyntax sieht wie folgt aus:

```
$ tagger/bin/separator-punctuation -h
USAGE: separate [-d "``.eos.."] [-D "``.para.."] [+1] [+s] [+m] [+1
<lexicon>] <input file> [<output file>]
```

```
-d <tag>: Use <tag> to mark end of sentence
-D <tag>: Use <tag> to mark end of paragraph
```

```
+l: One word per line output format
+s: Separate punctuation
+m: Print end of sentence and end of paragraph markers
+l <file>: Use the list of word forms ending with a period which are
stored in <file>
```

Fehlt <output file>, wird auf STDOUT geschrieben; fehlt auch <input file>, wird von STDIN gelesen. Im KoKS-Projekt lautete der Aufruf

```
$ tagger/bin/separate-punctuation +l +s +l tagger/lib/german-abbreviations
```

Es werden also Punkte abgerückt und alle Token auf eine eigene Zeile geschrieben; Abkürzungen werden mit einer Wortliste identifiziert.

### E.1.2 Tokenisierkorrektur: bin/metatag-correction

Der Tagger des IMS ignoriert SGML-Markup (in spitzen Klammern). Ein solches Markup-Tag geht jedoch beim Tokenisieren kaputt, wenn es Leerzeichen enthält; dann befindet sich die schließende Klammer nicht mehr auf derselben Zeile wie die öffnende, und der Tagger erkennt das Tag nicht als ein solches. Diesen Fehler korrigiert das Perl-Skript bin/metatag-correction: Es geht den Output des Tokenizers zeilenweise durch und fügt an jede Zeile, die mit einer spitzen Klammer beginnt, so lange nachfolgende Zeilen an, bis sie mit einer schließenden spitzen Klammer endet. Das Skript reiht sich in die Pipeline ein, liest also von STDIN und schreibt auf STDOUT.

### E.1.3 Sonderzeichenrekonstruktion: bin/umlaut.pl

Das Korpus der de-news verwendet keine deutschen Sonderzeichen (ä, ö, ü, ß), sondern reines ASCII. Für die Konvertierung in einen umfangreicheren Zeichensatz, etwa iso-latin1, wäre eine eindeutige Kodierung mit ASCII-Zeichen wünschenswert, wie sie etwa in HTML oder in  $\LaTeX$  üblich ist. Leider wird von dieser Möglichkeit im de-news-Korpus kein Gebrauch gemacht. Stattdessen werden Umlaute, wie es in bestimmten Bereichen des Schriftdeutschen (Formulare, Kreuzworträtsel etc.) üblich ist, durch ae, oe bzw. ue und das ß durch ss ersetzt. Das bereitet zwar nicht dem menschlichen, wohl aber dem maschinellen Leser Probleme: Der Tagger erkennt die Wörter nicht, da sie in seinem Lexikon anders – nämlich mit Sonderzeichen – eingetragen sind.

Um diese Problematik zu umgehen, bieten sich zwei Wege an: Das Lexikon des Taggers kann erweitert werden, sodass Wörter mit Umlaut oder ß auch in der entsprechenden Form ohne Umlaut und ß eingetragen sind. Alternativ muss der Text vor dem Taggen verändert werden: Die Sonderzeichen müssen rekonstruiert werden.

Die erste Möglichkeit ist mit einigem Aufwand verbunden: Das Lexikon liegt nicht als plain text vor, sondern ist in der (binären) Parameterdatei verborgen. Die Parameterdatei müsste also durch erneutes Training des Taggers mit erweitertem Lexikon neu erstellt werden. Einfacher ist die zweite Möglichkeit, für die wir uns auch entschieden: Da im KoKS-System mehrere Wörterbücher vorhanden sind (vgl. Kap. 4.1, kann die Sonderzeichenrekonstruktion mit Rückgriff darauf erledigt werden.

Dieses Vorgehen übernimmt das Perl-Skript bin/umlaut.pl: Es geht den bereits tokenisierten Text Token für Token durch. Enthält ein Wort ae, oe, ue oder ss, werden alle möglichen Alternativen gebildet. Diese Menge wächst exponentiell: Bei einem Vorkommen gibt es zwei Kandidaten für die korrekte Wortform, bei zwei Vorkommen bereits vier, bei dreien sind es acht Kandidaten usw. Diese Kandidaten werden – angefangen bei der Originalversion – in der Datenbank nachgeschlagen. Sobald ein Kandidat bekannt ist, ersetzt das Skript das Original durch diese Alternative und geht zum nächsten Token über.

Nun kommt es durchaus vor, dass ein Wort und sämtliche Alternativen unbekannt sind. In diesem Falle greift eine linguistisch basierte Korrektur. Sie ersetzt ae, oe, ue bzw. ss nach bestimmten Kriterien. Ae und oe bereiten wenig Probleme, denn sie werden fast immer durch ä bzw. ö ersetzt. Ue bleibt dagegen relativ oft ue (z.B. e nach Diphthong: Bauersfrau, Feuerleiter; e nach qu: Quelle). Auch ss ist kein einfacher Fall: Als Schärfungsmarkierung bleibt es oft ss, ß wird es nur am Wortende (Problem: Komposita wie „Naßzelle“); ss ergibt sich nicht selten auch an Morphemgrenzen (z.B. „Busstation“). Verschiedene Kriterien werden hier abgewogen und führen zu einem recht zuverlässigen Ergebnis.

Das Skript liest von STDIN und schreibt auf STDOUT. Ein- und Ausgabeformat sind reiner Text, wobei jede Zeile nur ein Token enthalten darf.

### E.1.4 Tagging und Lemmatisierung: `tagger/bin/tree-tagger`

Das Taggen und Lemmatisieren der tokenisierten Texte übernimmt der Tree-Tagger des IMS Stuttgart. Das IMS stellt neben dem Tagger an sich auch sprachspezifische Parameterdateien u.a. für Deutsch und Englisch zur Verfügung. Die Kommandosyntax sieht wie folgt aus:

```
USAGE: tree-tagger <parameter file> {<input file>} {<output file>}}
        {-eps <epsilon>} {-base} {-proto} {-sgml} {-token} {-lemma}
```

Hierbei bedeuten:

parameter file	sprachspezifische Parameterdatei
input file	zu taggende Datei; fehlt diese Angabe, liest der Tagger von STDIN
output file	Zieldatei; fehlt diese Angabe, schreibt der Tagger auf STDOUT
-sgml	Tagger ignoriert Token in spitzen Klammern
-token	gibt vor dem Tag das Token aus
-lemma	gibt nach dem Tag die Grundform aus
-base	Tagging erfolgt ohne Kontextinformation, nur auf Basis lexikal. Wissens
-proto	es wird eine Datei <code>lexicon-protocol.txt</code> angelegt, die zu jedem Wort die Ambiguität des Tags/Lemmas und das Quelllexikon nennt
epsilon	Wert, der (statt 0) für Wörter, die im Lexikon, nicht aber im Trainingskorpus enthalten sind; default ist 0,1

Für das KoKS-Projekt waren die Optionen `-sgml`, `-token` und `-lemma` aktiviert. Das Inputformat des Taggers ist Text, wobei jede Zeile als ein Token/Wort behandelt wird. Das Output-Format sieht wie folgt aus:

```
Token\tTag\tLemma
```

### E.1.5 Satzendenerkennung: `punkt-tagger-(german|english)`

Nicht immer markieren Punkte ein Satzende. Abkürzungen und Ordinalzahlen etwa werden ebenfalls durch eine Punkt terminiert, manchmal konvergieren auch Satzenden- und Abkürzungspunkt. Für den Satzaligner ist es unerlässlich, korrekte Sätze als Alignment-Einheiten zu erhalten. Die Perl-Skripte `bin/punkt-tagger-german` und `bin/punkt-tagger-english` versuchen, die Funktion der Interpunktionszeichen `'.`, `':`, `!`, und `?'` zu identifizieren.

Klassischerweise konkurrieren hier zwei Ansätze miteinander: eine regel- und eine statistikbasierte Erkennung. Die Information, die durch das Tagging vorliegt, führte dazu, dass die KoKS-Tools einen regelbasierten Ansatz verfolgen.

Die meisten Abkürzungen werden vom Tokenisierer durch die mitgeführte Abkürzungsliste erkannt; der Punkt wird dabei nicht abgerückt, die Buchstabenfolge samt Punkt also als ein Token behandelt. Hierbei übersieht der Tokenisierer aber womöglich, dass der Abkürzungs- zugleich ein satzbeendender Punkt sein kann. Das gleiche Problem tritt auch bei Ordinalzahlen auf. Zudem kann eine begrenzte Wortliste nicht alle Abkürzungen enthalten. Die Tools arbeiten mit verschiedenen Kriterien.

Das wichtigste ist die Nutzung des Taggergebnisses: Folgt auf eine Punkt ein großgeschriebenes Wort, das aber ein kleingeschriebenes Lemma besitzt, spricht dies eindeutig für ein Satzende. Dies wird nach jedem einzelnen Punkt (also nur `'.` als Token), aber auch nach Ordinalzahlen und Abkürzungen geprüft. Die Erfolgsquote liegt hier im Englischen höher als im Deutschen: Wörter werden im Englischen nur am Satzanfang (und bei Eigennamen) großgeschrieben, im Deutschen dagegen alle Substantive; das Deutsche verfügt somit über mehr großgeschriebene Lemmata.

Ein weiteres Kriterium versucht, unbekannte Abkürzungen zu identifizieren: Enthält das Token vor einem Punkt keinen Vokal, ist es wahrscheinlich eine Abkürzung. Zudem werden Abkürzungen im Deutschen i.d.R. dadurch gebildet, dass nach dem ersten Buchstaben der zweiten Silbe der Rest abgeschnitten wird. Dadurch ergeben sich am Ende von Abkürzungen Zeichenketten, die im Deutschen nicht als Endrand eines Wortes möglich sind (z.B. „Abk“ als Abkürzung für „Abkürzung“: „bk“ ist keine im Deutschen mögliche Zeichenfolge am Wortende). So können auch einige unbekannte Abkürzungen erkannt werden; ein ungültiger Endrand ergibt sich aber nicht in jedem Fall.

## E.2 Ausblick: Verbesserungsmöglichkeiten

Der Tokenisierer könnte „von Haus aus“ auf den Umgang mit SGML-Tags erweitert werden. Zudem scheint er einen Bug zu enthalten: Folgt auf einen Punkt nach dem Leerzeichen ein Umlaut, wird der Punkt nicht vom vorhergehenden Wort abgerückt. Dies sollte aber nicht schwierig zu beheben sein.

Bei der Sonderzeichenrekonstruktion bieten sich zahlreiche, aber allesamt recht aufwändige Verbesserungen an. Mit phonologischem und morphologischem Wissen können Wort-, Morphem- und Silbengrenzen erkannt und Umlaute dadurch besser rekonstruiert werden. Der Aufwand erscheint jedoch sehr hoch: Die Sonderzeichenproblematik betrifft nur das de-news-Korpus, und die meisten Fälle werden durch den Lookup in der Datenbank erschlagen. Verbesserungen sind also nur für einen kleinen Teil an Wörtern zu erwarten.

Der Satzdenenerkennung schließlich kann alternativ auf ein statistisches Verfahren umgestellt oder erweitert werden. In welchem Maße Verbesserungen des Ergebnisses zu erwarten sind, ist unklar. Gegebenenfalls kann ein statistisches Verfahren entwickelt werden, das auf getaggten Daten arbeitet und so auch Lemmata mit einbezieht.

## E.3 Die Tagsets

Hier finden sich Erläuterungen der einzelnen Tags. Es handelt sich vor allem um die Tags der vom Tagger verwendeten Tagsets; darüber hinaus sind hier aber auch KoKS-eigene Tags und Tags, die aus einem der genutzten Wörterbücher (dem LQL-Wörterbuch) extrahiert sind und die mit in das System eingeflossen sind. Zur tiefergehenden Lektüre sei nochmals auf die folgenden Publikationen verwiesen.

STTS-Tagset	<a href="ftp://www.ims.uni-stuttgart.de/pub/corpora/stts_guide.ps.gz">ftp://www.ims.uni-stuttgart.de/pub/corpora/stts_guide.ps.gz</a>
Penn-Treebank-Tagset	<a href="ftp://ftp.cis.upenn.edu/pub/treebank/doc/cl93.ps.gz">ftp://ftp.cis.upenn.edu/pub/treebank/doc/cl93.ps.gz</a>
	<a href="ftp://ftp.cis.upenn.edu/pub/treebank/doc/tagguide.ps.gz">ftp://ftp.cis.upenn.edu/pub/treebank/doc/tagguide.ps.gz</a>

### E.3.1 STTS (deutsch)

PoS-Tag	Beschreibung	Beispiele
ADJA	attributives Adjektiv	<i>[das] grosse [Haus]</i>
ADJD	adverbiales oder prädikatives Adjektiv	<i>[er fährt] schnell [er ist] schnell</i>
ADV	Adverb	<i>schon, bald, doch</i>
APPR	Präposition; Zirkumposition links	<i>in [der Stadt], ohne [mich]</i>
APPRART	Präposition mit Artikel	<i>im [Haus], zur [Sache]</i>
APPO	Postposition	<i>[ihm] zufolge, [der Sache] wegen</i>
APZR	Zirkumposition rechts	<i>[von jetzt] an</i>
ART	bestimmter oder unbestimmter Artikel	<i>der, die, das, ein, eine</i>
CARD	Kardinalzahl	<i>zwei [Männer], [im Jahre] 1994</i>
FM	Fremdsprachliches Material	<i>[Er hat das mit "] A big fish ["] übersetzt]</i>
ITJ	Interjektion	<i>mhm, ach, tja</i>
KOUI	unterordnende Konjunktion mit „zu“ und Infinitiv	<i>um [zu leben], anstatt [zu fragen]</i>
KOUS	unterordnende Konjunktion	<i>weil, dass, damit, mit Satz wenn, ob</i>
KON	nebenordnende Konjunktion	<i>und, oder, aber</i>
KOKOM	Vergleichspartikel, ohne Satz	<i>als, wie</i>
NN	normales Nomen	<i>Tisch, Herr, [das] Reisen</i>
NE	Eigennamen	<i>Hans, Hamburg, HSV</i>
PDS	substituierendes Demonstrativpronomen	<i>dieser, jener</i>
PDAT	attribuierendes Demonstrativpronomen	<i>jener [Mensch]</i>
PIS	substituierendes Indefinitpronomen	<i>keiner, viele, man, niemand</i>
PIAT	attribuierendes Indefinitpronomen ohne Determiner	<i>kein [Mensch], irgendein [Glas]</i>

Fortsetzung nächste Seite

PoS-Tag	Beschreibung	Beispiele
PIDAT	attribuierendes Indefinitpronomen mit Determiner	<i>[ein] wenig [Wasser], [die] beiden [Brüder]</i>
PPER	irreflexives Personalpronomen	<i>ich, er, ihm, mich, dir</i>
PPOSS	substituierendes Possessivpronomen	<i>meins, deiner</i>
PPOSAT	attribuierendes Possessivpronomen	<i>mein [Buch], deine [Mutter]</i>
PRELS	substituierendes Relativpronomen	<i>[der Hund,] der</i>
PRELAT	attribuierendes Relativpronomen	<i>[der Mann,] dessen [Hund]</i>
PRF	reflexives Personalpronomen	<i>sich, einander, dich, mir</i>
PWS	substituierendes Interrogativpronomen	<i>wer, was</i>
PWAT	attribuierendes Interrogativpronomen	<i>welche [Farbe], wessen [Hut]</i>
PWAV	adverbiales Interrogativ- oder Relativpronomen	<i>warum, wo, wann, worüber, wobei</i>
PAV	Pronominaladverb	<i>dafür, dabei, deswegen, trotzdem</i>
PTKZU	„zu“ vor Infinitiv	<i>zu [gehen]</i>
PTKNEG	Negationspartikel	<i>nicht</i>
PTKVZ	abgetrennter Verbzusatz	<i>[er kommt] an, [er fährt] rad</i>
PTKANT	Antwortpartikel	<i>ja, nein, danke, bitte</i>
PTKA	Partikel bei Adjektiv oder Adverb	<i>am [schönsten], zu [schnell]</i>
TRUNC	Kompositions-Erstglied	<i>An- [und Abreise]</i>
VVFIN	finites Verb, voll	<i>[du] gehst, [wir] kommen [an]</i>
VVIMP	Imperativ, voll	<i>komm [!]</i>
VVINFIN	Infinitiv, voll	<i>gehen, ankommen</i>
VVIZU	Infinitiv mit „zu“, voll	<i>anzukommen, loszulassen</i>
VVPP	Partizip Perfekt, voll	<i>gegangen, angekommen</i>
VAFIN	finites Verb, aux	<i>[du] bist, [wir] werden</i>
VAIMP	Imperativ, aux	<i>sei [ruhig !]</i>
VAINF	Infinitiv, aux	<i>werden, sein</i>
VAPP	Partizip Perfekt, aux	<i>gewesen</i>
VMFIN	finites Verb, modal	<i>dürfen</i>
VMINF	Infinitiv, modal	<i>wollen</i>
VMPP	Partizip Perfekt, modal	<i>[er hat] gekonnt</i>
XY	Nichtwort, Sonderzeichen enthaltend	<i>D2XW3</i>
\$,	Komma	<i>,</i>
\$.	Satzbeendende Interpunktion	<i>. ? ! ; :</i>
\$(	sonstige Satzzeichen; satzintern	<i>- [ ]()</i>

### E.3.2 Penn-Treebank-Tagset (englisch)

PoS-Tag	Beschreibung
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition/subord. conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural

Fortsetzung nächste Seite

PoS-Tag	Beschreibung
NP	Proper noun, singular
NPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PP	Personal pronoun
PP\$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol (mathematical or scientific)
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund/present participle
VBN	Verb, past participle
VBP	Verb, non-3rd ps. sing. present
VBZ	Verb, 3rd ps. sing. present
WDT	<i>wh</i> -determiner
WP	<i>wh</i> -pronoun
WP\$	Possessive <i>wh</i> -pronoun
WRB	<i>wh</i> -adverb
#	Pound sign
\$	Dollar sign
.	Sentence-final punctuation
,	Comma
:	Colon, semi-colon
(	Left bracket character
)	Right bracket character
``	Straight double quote
`	Left open single quote
"	Left open double quote
'	Right close single quote

### E.3.3 Eigene Tags und Annotationen

Zusätzlich zu den Tags, die der Tagger vergibt, liegen in den in KoKS getaggten Texten weitere Annotationen vor. Die Satzenerkennung vergibt ein eigenes Tag für als das Satzende markierende Interpunktion erkannte Satzzeichen (ein Punkt am Satzende wird mit SATZ-P getaggt). Die dadurch gewonnene textstrukturelle Information wird in Form von SGML-Tags vermerkt, da dies für den Aligner von Vorteil ist. Folgendes Beispiel macht dies deutlich:

---

```

beispiel-de.tag
Mein PPOSAT mein
Wochenende NN Wochenende
<SATZ>
<segmentgrenze>
<ABSATZ>
Letztes NN Letzte
Wochenende NN Wochenende
war VAFIN sein
langweilig ADJD langweilig
. SATZ-P .
<SATZ>
```

```

<segmentgrenze>
Die      ART      d
Fete     NN          Fete
zum      APPRART   zum
Ferienbeginn NN      Ferienbeginn
fiel     VVFIN      fallen
ins      APPRART   ins
Wasser   NN          Wasser
,        $,        ,
weil     KOUS      weil
die      ART      d
Disco    NN          Disco
abgebrannt VVPP      abbrennen
war      VAFIN      sein
.        SATZ-P   .
<SATZ>
<segmentgrenze>
Außerdem ADV      außerdem
kam      VVFIN      kommen
auch     ADV      auch
nichts   PIAT      nichts
Anständiges NN      Anständige
im       APPRART   im
Fernsehn NN      <unknown>
.        SATZ-P   .
<SATZ>
<segmentgrenze>
<ABSATZ>

```

Auf ein SATZ-P-PoS-Tag folgen die SGML-Tags <SATZ> und <segmentgrenze>. Ersteres markiert für den Aligner ein Satzende, Letzteres ein Segmentende. Vor dem Alignen tauchen diese SGML-Tags immer gemeinsam auf, nach dem Alignen kann ein Segment aber auch aus mehr als einem Satz bestehen (wenn etwa zwei deutsche Sätze zu einem englischen korrespondieren). Die <ABSATZ>-Markierung gewinnt KoKS bereits bei der Normalisierung (also vor dem Taggen) aus den Originaltexten, sofern dies daraus hervorgeht. Auch diese Information ist für den Aligner gedacht: Ein deutscher und ein englischer Text sollten die gleiche Anzahl von Absätzen haben, dann ist ein Satzalignment nur innerhalb von Absätzen nötig, was das Verfahren vereinfacht.

Hier noch einmal ein Überblick über die zusätzliche Annotation:

PoS-Tag	Beschreibung
SATZ-P	Satzbeendende Interpunktion; PoS-Tag
<SATZ>	Satzende; Markierung für Alignment
<segmentgrenze>	Delimiter für zu alignende/alignte Segmente; Markierung für Alignment
<ABSATZ>	Absatzende; Markierung für Alignment

### E.3.4 LQL-Tagset (deutsch & englisch)

PoS-Tag	Beschreibung
ADJ	Adjektiv
ADV	Adverb
ART	Artikel
IMP	Imperativ

*Fortsetzung nächste Seite*

<b>PoS-Tag</b>	<b>Beschreibung</b>
ITJ	Interjektion
KON	Konjunktion
NN	Nomen
PREP	Präposition
PRON	Pronomen
V	Verb
VI	intransitives Verb
VT	transitives Verb

## Anhang F

# Alignment – Dokumentation

### F.1 Überblick

Die Abbildung F.1 zeigt das schematische Zusammenspiel der Kommandozeilentools für das Alignment im KoKS-Projekt. Input für die Alignmentmaschinerie sind Paare von getaggtten parallelen Texten (z.B. `text1.de.tag` und `text1.en.tag`, Dateiformat siehe Anhang F.3.1), aus denen im nächsten Schritt Abstandsmatrizen erzeugt werden. Während `matrix.py` eine einzige große Matrix von allen Inputsätzen erzeugt, erzeugt `nmatrix.py` für jeden Absatz der Eingabetexte eine Matrix. Mit `mergemtr.py` können diese zu einer Gesamtmatrix zusammengefügt werden. Dieses zweite Vorgehen spart eine große Zahl von Abstandsberechnungen, da unter der Annahme, dass die Absätze bereits im Input korrekt align sind, Abstandsberechnungen zwischen Sätzen aus nicht korrespondierenden Absätzen überflüssig sind. Die nicht berechneten Bereiche werden mit dem schlechtesten Abstandswert gefüllt.

Eine so erzeugte Matrix kann nun über die Standardeingabe in die Alignmentkomponente `al.jar` gepiped werden. Dort wird mittels eines A\*-Suchalgorithmus der günstigste Alignment-Pfad durch die übergebene Matrix ermittelt und in einem speziellen Format als Segmentzuordnungen auf der Standardausgabe ausgegeben.

### F.2 Tools und Optionen

Die Tools des Aligners befinden sich alle im Verzeichnis `align` im CVS Baum.

#### F.2.1 Abstandsmass

Es gibt eine Kommandozeilenschnittstelle `align/abstand.py`. Das C&G-Mass ist nachimplementiert, Trigramm (TG) ist schnell, WB dagegen noch sehr langsam. TG2 verhält sich plausibler, wenn die Regionen unterschiedlich lang sind. WB2 ist ein verbesserte Reimplementation von WB.

```
$ abstand.py -i x.de.tag.al -j x.en.tag.al -m WB2 -u -r segment
```

Die Dateien `x.de—en.tag.al` enthalten getaggte und in Regionen unterteilte Dokumente. Die Abstandsausgabe erfolgt zusammen mit einer textuellen Darstellung der Region.

```
$ abstand.py -i a -j b -t
```

Die Dateien `a` und `b` enthalten zeilenweise alignen Text. `a` muss in der Quellsprache „de“, `b` in der Zielsprache „en“ sein.

```
$ abstand.py -t -c "Der Baum ist grün." -c "The tree is green."
```

Quell- und Zielsprachenzeilen koennen abwechselnd mit der Option `-c` angegeben werden.

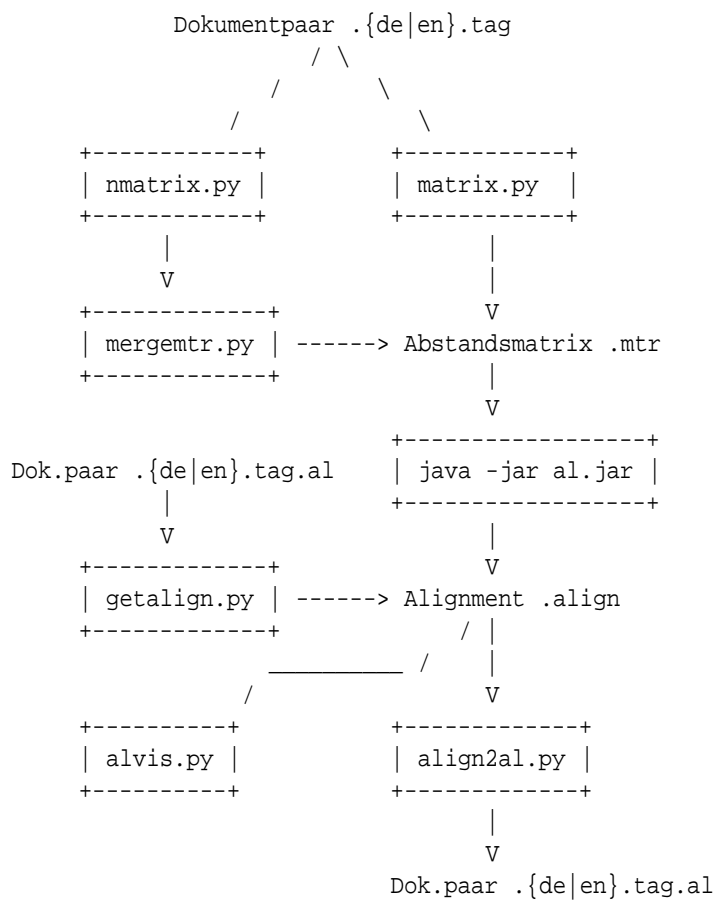


Abbildung F.1: Zusammenspiel der einzelnen Alignment-Tools

```
$ abstand.py -t -i c -m "C\&G" -s en -d de
```

Quell- und Zielsprachenzeilen stehen abwechselnd in der Datei c. Als Abstandsmass wird Church-und-Gale-Maß verwendet. Quell- und Zielsprache sind „en“ und „de“.

```
$ abstand.py -h
Usage: ./abstand.py <options>
options: -h          print out this help
          -m distmeas choose distance measure:
                    C&G  <AbstandChurchAndGale>
                    CG   <AbstandChurchAndGale>
                    EINS <Abstand immer 1.0>
                    NULL <Abstand immer 0.0>
                    TEST <AbstandTest>
                    TG   <AbstandTG>
                    TG2  <AbstandTG_2>
                    WB   <AbstandWB>
                    WB2  <AbstandWB_2 de nach en>
                    WB2N <AbstandWB_2 de nach en (nur Nomen)>
          -o filename write output to file
          -r regtype  one of
                    satz
                    segment
                    absatz
                    dokument
          -u          output index and text
          -v          verbose mode
          -.         dump current '-c' buffer
          -c string   use string as input line
          -d language set destination language
          -i filename read all input from file
          -j filename read dlang-input from file
          -l          input is a list of input filenames
          -s language set source language
          -t          tag input and treat lines as paragraphs
```

## F.2.2 Matrixausgabe

### matrix.py

Das Programm `matrix.py` berechnet zu einer Liste von Dokumenten (wahlweise Dateinamenspräfixe zu `.de.tag` und `.en.tag` Dateien oder Datenbank-IDs) eine Abstandsmatrix und schreibt sie zur Standardausgabe oder in Dateien. Das Abstands kann nicht per Kommandozeile gewählt werden sondern muss um Quellcode eingestellt werden. Ebenso fehlt die Möglichkeit, Matrizen absatzweise zu erzeugen.

### nmatrix.py

Das Programm `nmatrix.py` benutzt das Modul `eingabe.py` (siehe unten) zum Umgang mit dem zweisprachigen Eingabekorpus. Die Regionen, für die Abstandsmatrizen berechnet werden, können frei gewählt werden. Bedingung ist natürlich, dass die Anzahl in beiden Sprachen gleich ist. In Anlehnung an den Church-und-Gale-Aligner heißen sie Hardregions. Die Einheiten, die Zeilen und Spalten der Matrix repräsentieren, sind entsprechend Softregions. Die Dateiliste sollte nicht zu groß sein, da alle Dokumente vor der Zerlegung in Hardregions eingelesen werden.

```

$ nmatrix.py -h
Usage: ./nmatrix.py <options>
options:  -h          print out this help
          -l n        leave out cell which are more than n units away
                    from the conectiong line
          -L gradient modifies leave out distance
          -M maximum  modifies leave out distance
          -m distmeas choose distance measure:
                    C&G  <AbstandChurchAndGale>
                    CG   <AbstandChurchAndGale>
                    EINS <Abstand immer 1.01>
                    NULL <Abstand immer 0.0>
                    TEST <AbstandTest>
                    TG   <AbstandTG>
                    TG2  <AbstandTG_2>
                    WB   <AbstandWB>
                    WB2  <AbstandWB_2 de nach en>
                    WB2N <AbstandWB_2 de nach en (nur Nomen)>
          -o filename write output to file
          -O filename write abstandinfo to file
          -r regtype  soft regions: one of
                    satz
                    segment
                    absatz
                    dokument
          -R regtype  hard regions: one of
                    segment
                    absatz
                    dokument
                    korpus
          -u          output index and text
          -v          verbose mode
          -.          dump current '-c' buffer
          -c string   use string as input line
          -d language set destination language
          -i filename read all input from file
          -j filename read dlang-input from file
          -l          input is a list of input filenames
          -s language set source language
          -t          tag input and treat lines as paragraphs

```

### mergemtr.py

Das Programm `mergemtr.py` fügt mehrere Matrizen zu einer Gesamtmatrix zusammen. Die Matrizen werden blockweise diagonal aneinander gehängt. Die übrigen Zellen erhalten den Wert 1 (maximaler Abstand):

```

A A A A 1 1 1 1 1
A A A A 1 1 1 1 1
1 1 1 1 B 1 1 1 1
1 1 1 1 B 1 1 1 1
1 1 1 1 1 C C C 1
1 1 1 1 1 C C C 1
1 1 1 1 1 1 1 1 D

```

### F.2.3 Matrixalignment

#### mavis.jar

```
make dds
java -jar mavis.jar
```

#### al.jar

```
make dds
java -jar al.jar <abstandsmatrix.mtr >alignment.align
```

Mehrere Matrizen können in einem Aufruf verarbeitet werden, wenn die Anzahl vorher bekannt ist:

```
cat abstandsmatrix[1-5].mtr | java -jar al.jar 5
```

#### headingMapper.jar

headingMapper.jar erzeugt immer 1:1 Alignments. Es wird keine sequentielle Reihenfolge bevorzugt, insbesondere muss der Alignmentpfad nicht von oben links nach unten rechts in der Abstandsmatrix verlaufen. Die Ausgabereihenfolge ist auch beliebig.

#### alignment.py

alignment.py bestimmt die Anzahl der Matrizen und ruft dann al.jar auf. Gegenüber der Verwendung von mergemtr.py (s.o.) hat dies den Vorteil, dass nicht die Abstandsmatrix des gesamten Dokuments in den Speicher geladen werden muss. Da ein Modul fehlt, das die Satznummern der einzelnen Alignments, die von al.jar immer bezogen zum Absatzanfang ausgegeben werden, auf den Dokumentanfang unrechnet, kann alignment.py derzeit noch nicht sinnvoll eingesetzt werden. Des Weiteren muss zu einer guten Auslastung der Alignment-Pipe alignment.py noch so umgeschrieben werden, dass es jede Matrix an al.jar weiter reicht, sobald sie verfügbar ist, und auch das Ergebnis sofort ausgibt.

#### getalign.py

Das Programm getalign.py erstellt eine .align Datei aus bereits aligneten Dokumentpaaren .de.tag.al und en.tag.al oder aus ein in der Datenbank stehendes Dokumentpaar.

```
$ getalign.py -d 543
$ head db_543.align
Dokument-DB-ID 543
nach Satz
0      [0]    [0]
1      [1, 2] [1]
2      [3, 4] [2]
3      [5]    [3]
4      [6]    [4]
5      [7]    [5]
6      [8]    [6]
7      [9]    [7]
$
```

#### alvis.py

Liest eine .align Datei und die zugehörigen Dokumente ein und stellt das Alignment textuell dar.

```

$ alvis.py db_543.align
Dokument-DB-ID 543
--- Segment Nr. 0 Alignment [0] - [0] ---
Europäischer Sozialfonds : Die Kommission nimmt ein Maßnahmenpaket für Belgien a
European Social Fund : the Commission adopts a series of measures to promote emp

--- Segment Nr. 1 Alignment [1, 2] - [1] ---
Im Rahmen dieses Ziel-3-Programms(2 ) werden 69 Mio. EUR für Maßnahmen zur sozia
This programme , which is valued at 69 million euros and comes under Objective 3
...

```

## F.2.4 Erzeugen alignter Dateien

### align2al.py

Von al.jar erzeugte, lineare, ganze Dokumente umfassende Alignments können mit align/align2al.py in .de.tag.al/en.tag.al Dateien geschrieben werden. Angegeben wird nur die .align Datei. '-' liest von STDIN.

```

$ align2al.py -h
Benutzung: ./align2al.py [-t] <.align-Datei>
           -t = Ausgabe ins Arbeitsverzeichnis

```

## F.2.5 Dokumentausgabe

### dokvis.py

Das Kommandozeilentool dokvis.py kann die Sätze, Segmente und Absätze eines Dokuments anzeigen.

```

$ dokvis.py
Benutzung: ./dokvis.py [<Optionen>] <getaggttes Dokument>
oder       : ./dokvis.py [<Optionen>] <db-id> <sprache>
Optionen : -r Regionentyp  Werte: satz, segment, absatz

```

### eingabe.py

Das Modul eingabe.py bietet eine ähnlich Schnittstelle für Dokumentpaare, z.B.

```

$ ls ../korpus/de-news/1997/05/de-news-1997-05-???.?.tag.al >list
$ python eingabe.py -n -l -r dokument -i list >list.out

```

```

$ python eingabe.py -h
Usage: eingabe.py <options>
options:  -h          prints this help
          -n          alternate docs in output
          -r regtype  soft regions: one of
                   satz
                   segment
                   absatz
                   dokument
          -v          verbose
          -.         dump current '-c' buffer
          -c string   use string as input line
          -d language set destination language
          -i filename read all input from file
          -j filename read dlang-input from file

```

```

-l          input is a list of input filenames
-s language set source language
-t          tag input and treat lines as paragraphs

```

## F.2.6 Aligner von Church und Gale

Der Aligner hat Probleme mit leeren Regionen und Regionen, die ohne Delimiter enden, da er nicht zeilenweise parst sondern die Eingabe an den Delimitern zerlegt und dabei nur die als Argument auf der Kommandozeile übergebenen Delimiter `<segmentgrenze>` und `<ABSATZ>` berücksichtigt.

Den Aligner ist im CVS-Baum unter `align` verfügbar und kann mit dem Makefile im gleichen Verzeichnis übersetzt und in `bin` installiert werden.

```

$ make align
$ align -D '<ABSATZ>' -d '<segmentgrenze>' file1 file2

```

erstellt zwei Dateien: `file1.al` und `file2.al`. Die Anzahl der Zeilen `<ABSATZ>` muss in beiden Eingabedateien gleich sein.

Es wurden am Quellcode einige Änderungen vorgenommen. Z.B. kann das ursprüngliche Format für Ein- und Ausgabe mit der Option `-e` erzwungen werden. Dann wird nur der Text bis zum letzten Harddelimiter verarbeitet und innerhalb der Hardregion nur der Text bis zum letzten Softdelimiter. `align -e` erwartet also, dass jede Region mit einem Delimiter endet. Das ist das ursprüngliche Verhalten des Aligners. Z.B. würde in der Eingabe

```

A
<ABSATZ>
B

```

der Text B nicht verarbeitet werden. Analog gilt, wenn z.B. Text A = „a1 `<segmentgrenze>` a2“, B == „b1“, d.h. Eingabe

```

a1
<segmentgrenze>
a2
<ABSATZ>
b1

```

dass a2 ignoriert wird, weil innerhalb der Region A auf a2 kein Delimiter folgt. Die `<ABSATZ>`-Zeilen sind nicht Bestandteil der Regions. Sie trennen sie. In diesem Verarbeitungsmodus muss also in der Eingabe jeder Satz mit `<segmentgrenze>` und jeder Absatz mit `<ABSATZ>` enden.

Da in der Vorverarbeitung diese zusätzlichen `<ABSATZ>` und `<segmentgrenze>` - Zeilen in der ersten Hälfte der Projektzeit nicht eingefügt wurden, wurde der Aligner so umgeschrieben, dass er auch den Text nach dem letzten Delimiter einer Region als Subregion akzeptiert, so dass im obigen Beispiel auch a2 und b1 erfasst werden. Nachteil: Steht am Ende doch ein Delimiter, dann wird dies als eine leere Region interpretiert. Auf Segmentebene sind leere Regions notwendig. Und auch auf Absatzebene mag es manuell oder später auch automatisch (Absatzalignment) eingefügte leere Absätze geben.

## F.3 Dateiformate

Rund um dem Aligner spielen einige Dateiformate eine wichtige Rolle. Sie geben an, wie Dokumente, Abstandsmatrizen, Alignments und sonstige Informationen kodiert werden. Werden die Dateiformate eingehalten, lassen sich einzelne Komponenten des Aligner austauschen oder für andere Zwecke verwenden.

```

18 jobs follow
Absatz 0 von Dokument-Datei temp/1997-05-19
nach satz
1      1
0.0
Absatz 1 von Dokument-Datei temp/1997-05-19
nach satz
5      5
0.37228732239986267
0.68828350812593986
...

```

Abbildung F.2: Mehrere Matrizen in einer Datei

### F.3.1 Dokument

Das zu alignende Dokumentpaar muss in Dateiform vorliegen. Eine Datei enthält das Dokument in der Sprache Deutsch, eine zweite Datei enthält es in Englisch. Die Dokumente sollten dem Ausgabeformat des KoKS-Taggers entsprechen.

Jede dreispaltige Zeile ist ein Atom bestehend aus Wort, POS-Tag und Lemma. Die Delimiterzeilen <SATZ>, <segmentgrenze>, <ABSATZ> und <DOKENDE> markieren das Ende der jeweiligen Region. <DOKENDE> darf weggelassen werden. Zwischen <KOKS> und </KOKS> werden alle Zeilen ignoriert. Das Verhalten ist allerdings nicht definiert, wenn der Bereich eine Delimitergrenze überspannt. Im Programm korpus2db.py wird die Eingabe zuerst an den Delimitern getrennt. Die Reichweite von <KOKS> ist dann auf einen Satz beschränkt. Im Aligner haben dagegen nur die Delimiter <DOKENDE> und <KorpusEnde> vorrang. Dann werden die <KOKS>-Markierungen ausgewertet, wobei Schachtelungen erlaubt sind. Die übrig bleibenden Delimiter bestimmen schließlich die weitere Struktur.

### F.3.2 Abstandsmatrix

Ürsprünglich war vorgesehen, dass für jedes zu alignenden Dokumentpaar nur eine Abstandsmatrix erstellt wird. Später wurde aber ein absatzweises Alignen notwendig, so dass für jedes Absatzpaar eine Abstandsmatrix benötigt wird. Da die vorhandenen Tools dokumentweise arbeiten, müssen mehrere Matrizen in einer Datei gespeichert werden können. Es werden daher zwei Dateiformate für Abstandsmatrizen unterschieden.

#### Einzelmatrix

Die ersten drei Zeilen der Datei bilden den Dateikopf. Danach folgen durch Whitespace getrennt die Matrixeinträge. Die erste Zeile beschreibt das zugrundeliegende Regionenspaar. Sind es Dokumenten von Dateien, deren Dateinamen auf .de.tag und .en.tag enden, dann reicht es, nur den Dateistamm anzugeben. Beispiele:

```

Dokument-Datei ./de-news/1997/1997-05-19
Paar Dokument-Datei ./eu/97/567-de.tag und Dokument-Datei ./eu/97/567-en.tag
Dokument-DB-ID 432
Absatz 5 von Dokument-Datei temp/test
Absatz 5 von Paar Dokument-Datei t2-de.tag und Dokument-Datei t2-en.tag

```

Die zweite Zeile gibt ab, welchem Regionentyp die Zeilen und Spalten der Matrix entsprechen. Hier steht i.d.R. nach Satz oder nach Absatz. Die dritte Zeile gibt Breite und Höhe der Matrix an.

```

Paar Dokument-Datei ./eu/97/567-de.tag und Dokument-Datei ./eu/97/567-en.tag
nach satz
0      [0]          [0,1]
1      [1]          [2]
2      [2]          [3]
3      [3]          [4]
4      [4]          [5]
5      [5]          [6]
6      [6]          [7]
7      [7,8,9,10,11]      [8,9,10,11,12]
8      [12]         [13]
9      [13]         [14]
10     [14]         [15]
11     [15]         [16]
12     [16]         [17]
13     [17]         [18]
14     [18,19]      [19,20,21]

```

Abbildung F.3: Korrekte Alignmentdatei

### Mehrere Matrizen

Die erste Zeile gibt an, wie viele Matrizen in der Datei enthalten sind. Dann folgen direkt hintereinander Einzelmatrizen wie oben beschrieben. Abbildung F.2 zeigt den Anfang einer Beispieldatei.

### F.3.3 Alignment

Das Alignmentformat beschreibt die Zuordnung der Sätze des deutschen Dokuments zu den Sätzen des englischen Dokuments. Das Dokumentpaar wird dabei so in Segmente unterteilt, dass folgende drei Bedingungen erfüllt sind.

- Jeder Satz der beiden Dokumente gehört zu genau einem Segment.
- Für jede Sprachseite müssen die Sätze eines Segments zusammenhängend sein.
- Die Reihenfolge der Sätze muss erhalten bleiben.

Eine Alignmentdatei hat den gleichen zweizeiligen Kopf wie eine einzelne Abstandsmatrix. Dann folgt die Beschreibung der Segmente. Jeder Zeile enthält durch Tabulatoren getrennt eine Segmentnummer, den Indexbereich der Sätze des deutschen Dokuments und den Indexbereich der Sätze des englischen Dokuments. Die Indexbereiche stehen in eckigen Klammern und enthalten kommagetrennt die Satznummern. Satz- und Segmentnummern beginnen bei 0.

Abbildung F.3 zeigt ein Alignment, das jeweils ein 1:2, 2:3 und 5:5 Alignment und zwölf 1:1 Alignments enthält.

### F.3.4 Abstandinfo

Wenn die beiden beteiligten Dokumente  $n$  und  $m$  Sätze haben, dann besteht die Abstandinfo-Datei aus  $nm$  Textblöcken, die jeweils mit der Zeile <SATZ> enden. Der Inhalt hängt von dem verwendeten Abstandsmaß ab.

## F.4 Schnittstellen

Das modulare Konzept des Aligner erfordert viele Schnittstellen. Zum einen müssen die in F.3 beschriebenen Dateiformate einheitlich verarbeitet werden. Dann sollen verschiedene Abstandsmaße definiert werden können, die alle auf die gleiche Weise benutzt werden. Schließlich stehen mehrere Alignmentoptimierer zur Verfügung, die mit Hilfe der Abstandsmatrix ein Alignment berechnen.

### F.4.1 Dokument

Das Python-Paket `Region`, das sich im Verzeichnis `align` befindet, erlaubt einen einfachen Zugriff auf die getaggtten Korpora.

```
from Region import dokument
dok = dokument.DokumentVonDatei('/home/koks/korpus/eu/97/567-de.tag')
for absatz in dok:
    print absatz
```

Ebenso können alignte Bitexte verarbeitet werden:

```
from Region import dokument
dok = dokument.DokumentPaarVonDateien('/home/koks/korpus/eu/97/567-de.tag',
'/home/koks/korpus/eu/97/567-en.tag', 'de', 'en')
for absatz in dok:
    print '[absatz, absatz.alignedRegion()]'
    print absatz.quelleninfo()
```

Im Abschnitt F.5.1 werden einige Klassen des `region`-Moduls beschrieben.

Das Python-Modul `eingabe` bietet eine einheitliche auf das `region`-Modul basierende Schnittstelle, um von der Kommandozeile aus eine Liste von Dokumentpaaren anzugeben. Details können den Programmen entnommen werden, die das Modul bereits verwenden.

Achtung: Beim Eintragen in die Datenbank wird ein anderer Dokumentparser verwendet, da dieser Teil sehr früh entstand. Hier gibt es Unterschiede bezüglich leere Regionen und fehlenden Delimiter. `Mavis` enthält noch einen dritten sehr einfach gehaltenen Dokumentparser, um die Satzinfos anzeigen zu können. Da aber im Tagger nach der Satzenderkennung eine sehr robuste Delimiterstruktur erzwungen wird, sollten keine Probleme auftreten.

### F.4.2 Abstandsmaß

Im Python-Modul `abstand` wird die Klasse `Abstand` definiert, die als Grundlage für alle Abstandsmaße dient. Der Abstand zwischen zwei beliebigen Regionen kann ermittelt werden, indem ein Objekt der Klasse `Abstand` mit den Regionen als Argumente gerufen wird. Alternativ kann die Funktion zwischen benutzt werden.

Mit der Funktion `setAiouT` kann ein Ausgabestrom für die `AbstandInfo` eingestellt werden.

### F.4.3 Alignmentoptimierer

Eigene Alignmentoptimierer können als Java-Klassen implementiert werden.

### F.4.4 Ausblick

#### Dynamische Regionenhierarchie

Die Regionenhierarchie ist statisch. Sie wird aus einer Datei geladen oder ausgehend von den Atomen konstruiert. Zur Laufzeit können aber nicht z.B. zwei Segmente verschmolzen werden. Zwar kann man die Satzlisten der beiden Segmente benutzen, um ein neues Segment zu bauen. Aber das neue Segment kann nicht die beiden Segmente im übergeordneten Absatz ersetzen. Python bietet dazu eine Schnittstelle, die im Modul `Region.region` noch zu implementieren ist. Dann könnte man Segmente in einem Absatz wie folgt verschmelzen:

```
# a sei ein Absatzobjekt
a[0:1+1] = Segment.segment(list(a[0])+list(a[1]))

# Wenn __add__ in region implementiert wird, dann geht es noch einfacher:
a[0:1+1] = Segment.segment(a[0]+a[1])
```

Eine dynamische Regionenhierarchie könnte sinnvoll beim Alignment verwendet werden.

### Abstandsmaß

Das Relevant-Objekt, das in der Klasse AbstandWB\_2 verwendet wird, sollte besser über eine Methode kommen, damit durch Ableiten WB2-Varianten erstellt werden können.

## F.5 Konzeption

### F.5.1 Dokument

Dokumente können mit den Modulen in align/Region und align/DatabaseAPI in Absätze, Segmente (align), Sätze und Atome zerlegt werden. Jede Region verhält sich wie eine Liste, die die entsprechenden Subregionen enthält. Dies gilt natürlich nicht für Atome. Alignte Dokumentpaare enthalten als Listenelemente alignte Absatzpaare, die wiederum alignte Segmentpaare enthalten. Es spielt keine Rolle, ob die Dokumente in einer KoKS-Datenbank oder in Dateien stehen.

Hier kurz ein paar Worte zur Klassenhierarchie:

**Region** Dies ist die Wurzel der Klassenhierarchie. Die Klassen Region führt die Funktionen getWortliste(), getTagliste(), getLemmaliste() und getText() ein. Letztere soll einen String liefern, der zur Darstellung der Oberflächenform der Region benutzt werden kann. Derzeit werden einfach die Wörter durch Whitespace getrennt geliefert. (S.u.) Noch zu implementieren ist, dass die Satzzeichen an das vorangehende Wort gerückt werden. Dies sollte in der Klasse Satz geschehen.

**Regionen** Die Klassen Regionen erbt von Region. Konstruktor erhält Liste von Region-Objekten. Die Objekte zeigen Listen-Verhalten: len(), \_getitem\_() ('getItemAtIndex()'). getText() liefert leerzeichengetrennt den Inhalt der Subregionen. getRegionenMit(class) liefert ein Regionen-Objekt, dessen Listenelemente von der gewünschten Klasse sind. Das ist nicht trivial, wenn z.B. eine Dokument-Region vorliegt und Segmente gewünscht sind.

**RegionenMit** Die Klassen RegionenMit erbt von Regionen. Konstruktor erhält Regionen-Objekt und gewünschte Regionen-Klasse Die Klasse überschreibt das Listen-Verhalten, damit die Listenelemente der RegionenMit-Objekte von der gewünschten Klasse sind. Z.B. ist RegionenMit(DokumentVonDb(dbconn, 123, 1), atom) eine Liste der Atome des Dokuments 123 aus der Datenbank (in der Sprache 1).

**Atom** Die Klassen Atom erbt von Region. Der Konstruktor erhält drei Strings: Wort, Tag und Lemma.

**AtomVonZeile** Die Klassen AtomVonZeile erbt von Atom. Konstruktor erhält tabulatorgetrennt Wort, Tag und Lemma. Whitespace nach Lemma wird eliminiert.

**Segment** Die Klasse Segment erbt von Regionen. Typischerweise übergibt man dem Konstruktor eine Liste von Satz-Objekten.

**SegmentVonZeilenliste** Die Klasse SegmentVonZeilenliste erbt von Segment. Konstruktor erhält eine Liste von Strings, die wie bei AtomVonZeile beschrieben aufgebaut sind.

**Absatz** Die Klasse Absatz erbt von Regionen. getText() hängt zusätzlich ein Newline an den String.

**AbsatzVonZeilenliste** Die Klasse `AbsatzVonZeilenliste` erbt von `Absatz`. Der Konstruktor erhält eine Liste von Strings. Die Liste wird bei den Strings, die mit `<segmentgrenze>` beginnen, zerschnitten und dann aus jedem Abschnitt ein Segment mit `SegmentVonZeilenliste` erzeugt.

**Dokument** Die Klasse `Dokument` erbt von `Regionen`. `getText()` tilgt Leerzeichen vor Newlines. (`Absatz.getText()` schließt den String mit einem Newline ab. `Regionen.getText()` trennt die Absätze eines Dokuments mit Leerzeichen. Diese sind überflüssig und unschön und werden daher von `Dokument.getText()` wieder entfernt. Alternativ könnte man sie durch z.B. drei Leerzeichen ersetzen, sodass alle Absätze bis auf den ersten des Dokuments eingerückt sind.

**DokumentVonDatei** Die Klasse `DokumentVonDatei` erbt von `Dokument`. Konstruktor enthält einen Dateinamen. Die Datei wird an Zeilen, die mit `<ABSATZ>` beginnen, zerschnitten und dann aus jedem Abschnitt ein Absatz mit `AbsatzVonZeilenliste` erzeugt. Des Weiteren signalisiert `<KOKS>`, dass zu ignorierende Zeilen folgen.

Anmerkung: Weniger Speicherintensiv wäre es, `AbsatzVonDateiImDateizeigerbereich(file, startzeile, endezeile)` usw. zu benutzen. Da aber geplant war, die Dokumente über die Datenbank zu verarbeiten, wurde dies nicht implementiert.

**AlignedRegion** Die Klasse `AlignedRegion` führt ein, dass eine Region eine zu ihr alignte Region hat. Neue Funktionen sind `alignedRegion()`, `getQuellsprache()` und `getZielsprache()`.

## F.5.2 wichtige Java-Klassen des Aligners (für `mavis.jar`, `al.jar`, `headingMapper.jar`)

Die folgenden Klassen stellen den Kern der Programme `mavis.jar`, `al.jar` und `headingMapper.jar` dar. Titel vielleicht um "wichtige Konzepte und Klassen" ergänzen...

- Die Schnittstelle zum Aligner besteht aus einer Abstandsmatrix. Die Klasse `de.denkselbst.matrix.Matrix` repräsentiert eine solche Abstandsmatrix und kann mit einer `.mtr`-Datei initialisiert werden. In dem GUI-Tool Mavis gibt es viele Objekte, die sich für Änderungen (Transformation der Daten, neue Datei geladen) interessieren, daher wurde ein einfaches Observer-Modell geschaffen (`package de.denkselbst.beobach` das vom Matrix-Objekt unterstützt wird).
- Die Klasse `de.denkselbst.MatrixAligner` schreibt vor, wie sich Objekte verhalten, die einen Alignmentpfad in einer Matrix ermitteln. Im Einsatz ist dabei hauptsächlich die von `de.denkselbst.MatrixAligner` abgeleitete Klasse `de.denkselbst.AStarMatrixAligner`, die den A\*-Suchalgorithmus zur Ermittlung eines Alignmentpfades benutzt. Andere Unterklassen von `MatrixAligner` implementieren weitere Suchverfahren (\*\* siehe javadoc, vor allem `HeadingMapper`, da und da \*\*). Für das GUI-Tool Mavis werden von `de.denkselbst.MatrixAligner` Methoden bereitgestellt, das Alignment in einem separaten Thread auszuführen. Für Kommandozeilentools wird diese Funktionalität nicht benötigt. Alignments werden in dieser Version als Pfad-Objekte abgeliefert. Später in der Entwicklung wurde die Klasse `de.denkselbst.matrix.KoksZuordnung` geschaffen, die Alignments adäquater repräsentiert. Pfade lassen sich in jedem Fall in `KoksZuordnungen` umwandeln.
- Ein Pfad `de.denkselbst.matrix.Pfad` ist eine Liste von Koordinatenpaaren. Im Rahmen von Graphensuchalgorithmen ist das ein geeignetes Mittel, um einen Pfad in einer Matrix zu beschreiben. Längere horizontale Abschnitte in einem Pfad werden als `n:m` - Alignments aufgefasst, durch eine Konvertierung des Pfades in eine `KoksZuordnung` wird dem Rechnung getragen. `MatrixAligner` liefern ihre Ergebnisse derzeit als Pfade.
- Diese Klasse `de.denkselbst.matrix.KoksZuordnung` repräsentiert ein Alignment. Ein Alignment besteht aus Zuordnungen (`KoksZuordnung.zuordnung`), die `n` Sätze aus `L1` mit `m` Sätzen aus `L2` verbinden. Eine Liste solcher Zuordnungen, die sequentiell die Menge aller Sätze bedeckt, ist eine `KoksZuordnung`. `KoksZuordnungen` können im Wagner-Format (\*\*siehe \*\*) textuell ausgegeben werden.
- Die Klasse `de.denkselbst.kommandozeile.Align` stellt einen Rahmen dar, in dem die oben genannten Objekte ins Leben gerufen werden. Die Main-Methode liest Matrizen von STDIN und

schreibt Alignments nach STDOUT. Diese Arbeit kann mit den oben beschriebenen Objekten erledigt werden. Die wichtigsten Schritte sind es, den Eingabestrom zu besorgen, ein Matrixobjekt und ein damit verbundenes MatrixAlignerobjekt zu erstellen, die Matrix aus STDIN zu füllen, zu normalisieren, den MatrixAligner das Alignment (als Pfad) finden lassen, den Pfad in eine KoksZuordnung umzuwandeln und das Ergebnis auf STDOUT auszugeben. Der folgende Code-Auszug zeigt, wie es geht:

```
// STDIN-Strom besorgen
BufferedReader puff = new BufferedReader(new InputStreamReader(System.in));

// Matrixobjekt anlegen
Matrix matrix = new Matrix();

// Alignerobjekt anlegen
AStarMatrixAligner aligner = new AStarMatrixAligner(matrix);

// besorge Matrix aus Stdin, Aligner bekommt die Matrixänderung
// automatisch von der Matrix mitgeteilt (Observermodell)
matrix.setzeDatenAusGepuffertemLeser(puff);

// Normalisiere Matrix
matrix.normalisiereDich();

// aligne
//System.err.println("starte Alignment");
Pfad p = aligner.align();

// schreibe Ergebnis
System.out.print(p.toWagnerString(matrix.holeQuelle(),matrix.holeInfo()));
```

Da es offensichtliche Performanzvorteile gibt, wenn nicht für jede Matrix einmal die JAVA-VM gestartet werden muss, können align mehrere Aufträge pro Aufruf erteilt werden. Dazu muss der erste Parameter ein Integer-Wert sein, der die Anzahl der Aufträge enthält. Um den obigen Code-Ausschnitt herum existiert eine Schleife. Wird align ohne Angabe der Auftragsanzahl aufgerufen, wird diese Schleife nur einmal durchlaufen.

### F.5.3 Das Paket `de.denkselbst.matrixAligner`

Enthaltene Klassen:

- `de.denkselbst.matrixAligner.AlignmentObserver`
- `de.denkselbst.matrixAligner.AlignmentObservable`
- `de.denkselbst.matrixAligner.MatrixAligner`
- `de.denkselbst.matrixAligner.LMEDMatrixAligner`
- `de.denkselbst.matrixAligner.BFSMatrixAligner`
- `de.denkselbst.matrixAligner.BFSMatrixAligner.Knoten`
- `de.denkselbst.matrixAligner.BFSMatrixAligner.Weg`
- `de.denkselbst.matrixAligner.BFSMatrixAligner.Agenda`
- `de.denkselbst.matrixAligner.BestCellMatrixAligner`
- `de.denkselbst.matrixAligner.AStarMatrixAligner`
- `de.denkselbst.matrixAligner.AStarMatrixAligner.ConfigDialog`

**de.denkselbst.matrixAligner.AlignmentObserver**

Es handelt sich um ein Interface, das die Methoden festlegt, die ein Objekt beherrschen muss, wenn es Alignment in einem anderen Thread lostreten will und über den Fortschritt benachrichtigt werden will. (Observer-Muster)

Methoden:

**public void alignmentFertig(de.denkselbst.matrix.Pfad, boolean)** diese Methode wird vom observierten MatrixAligner-Objekt aufgerufen, wenn der Alignmentprozess ein Ende gefunden hat. @param erfolg enthält einen booleschen Wert, der andeutet, ob das Unternehmen erfolgreich war, wenn ja, dann enthält @param p den Alignmentpfad.

**public void statusInformation(java.lang.String, boolean)** Übermittelt Statusinformationen (z.B. zu Ausgabe in einer grafischen Oberfläche)

**de.denkselbst.matrixAligner.AlignmentObservable**

Dieses Interface legt die Methoden fest, die ein Alignment-Objekt beherrschen muss, wenn es Alignment in einem eigenen Thread anbieten will. (Observer-Muster)

Methoden:

**public void align(de.denkselbst.matrixAligner.AlignmentObserver)** Starte Alignment unter Beobachtung/Einfluss von @param obs

**public void cancel()** AlignmentObserver kann durch Aufruf dieser Methode den Alignmentvorgang abbrechen

**de.denkselbst.matrixAligner.MatrixAligner**

Hierarchie:

java.lang.Object

Diese Klasse ist Superklasse aller Klassen, die einen Alignment-Pfad für eine Matrix ermitteln wollen, können oder sollen. Die Klasse MatrixAligner schreibt vor, wie sich Objekte zu verhalten haben, die einen Alignmentpfad in einer Matrix ermitteln wollen. Für das GUI-Tool Mavis wurden Methoden bereitgestellt, das Alignment in einem separaten Thread auszuführen. Für Kommandozeilentools wird diese Funktionalität nicht benötigt. Alignments werden in dieser Version als Pfad-Objekte abgeliefert. Später in der Entwicklung wurde die Klasse KoksZuordnung geschaffen, die Alignments adäquater repräsentiert. Pfade lassen sich in jedem Fall in KoksZuordnungen umwandeln. Soll das Alignment in einem eigenen Thread ablaufen, so ist die Methode public void align(AlignmentObserver obs) aufzurufen. Dabei ist zu beachten, dass dieser Aufruf pro Alignerobjekt nur einmal stattfinden darf, da sonst der zweite Aufruf den ersten Observer überschreibt. Also: Alignerobjekt instantiiieren, einmal alignen, dann Alignerobjekt wegwerfen. Anschließend neues Alignerobjekt machen, alignen, wegwerfen, usw.

Methoden:

**public void setzeMatrix(de.denkselbst.matrix.Matrix)** setzt eine neue Matrix ein

**public java.lang.String alignerInfo()** Liefert den alignerIdentString. Der kann dann z.B. im GUI als Fenstertitel gesetzt werden.

**public void konfigurationsDialogAbfahren()** Soll in Unterklassen so überschrieben werden, dass ein modaler Konfigurationsdialog abgefahren wird und die Einstellungen dann gesetzt werden. Bietet der Aligner keine Einstellmöglichkeiten, ist nichts weiter zu tun. Diese Methode wird von der grafischen Oberfläche (Mavis/MACV) aufgerufen, nachdem der Aligner instantiiert wurde und bevor das Alignment gestartet wurde.

**public de.denkselbst.matrix.Pfad align()** ermittelt (den) Alignment-Pfad (ist von den Unterklassen zu implementieren)

- public void align(de.denkselbst.matrixAligner.AlignmentObserver)** Diese Methode startet das Alignment in einem neuen Thread. Das Alignment-Ergebnis wird per Observer-Mechanismus an den AlignmentObserver obs zugestellt. (aus interface AlignmentObservable)
- public void cancel()** Diese Methode setzt die Flagge zum Beenden des Alignments (aus interface AlignmentObservable)
- public void abmelden()** Meldet den MatrixAligner als Beobachter von der Matrix ab und vergisst seinen Observer. „abmelden“ sollte das GUI zum MatrixAligner sagen, bevor es ihn entsorgt, da die Matrix den MatrixAligner sonst noch als Beobachter registriert hat, es also noch eine Referenz gibt und der Aligner folglich nicht vom Garbage-Collector eingesammelt werden kann!!!
- public void run()** run-Methode für Arbeit in einem eigenen Thread
- public void beobachtetesObjektHatSichGeregt(java.lang.Object, java.lang.String)** Auf Änderungen in der Matrix reagieren die Aligner sehr empfindlich, nämlich mit Abbruch des Alignments. (aus interface Beobachter)
- protected void informObserver(java.lang.String, boolean)** Convenience-Methode: Status-Nachricht an Observer (prüft auf observer != null)
- protected void informObserverZZZ(java.lang.String, boolean)** Convenience-Methode: Status-Nachricht an Observer, aber nur jede 100. Nachricht wird tatsächlich weitergereicht. (prüft auf observer != null)
- protected void informObserverAboutFinishedAlignment(de.denkselbst.matrix.Pfad, boolean)** Convenience-Methode: Alignmentergebnis an Observer weitergeben (prüft auf observer != null)

#### de.denkselbst.matrixAligner.LMEDMatrixAligner

Hierarchie:

```
java.lang.Object
  de.denkselbst.matrixAligner.MatrixAligner
```

Dieser Aligner basiert auf dem Levenshtein-Minimum-Edit-Distance Algorithmus. Er bietet eine bessere Laufzeit als der AStarMatrixAligner, aber liefert auch schlechtere Ergebnisse. Er entstand recht früh und in Zusammenarbeit mit Philip Reuter.

Methoden:

- public double get(int, int)**
- public int getXDim()**
- public int getYDim()**
- public de.denkselbst.matrix.Pfad align()** Align-Methode

#### de.denkselbst.matrixAligner.BFSMatrixAligner

Hierarchie:

```
java.lang.Object
  de.denkselbst.matrixAligner.MatrixAligner
```

Dieser MatrixAligner verwendet eine Best-First-Suchstrategie. Die hier vorliegende Implementierung diene vor allem dem Herumexperimentieren mit Strafen und eignet sich nur für kleine Matrizen. Der AStarMatrixAligner ist schneller und benötigt weniger Speicher, liefert aber im Standartfall das gleiche Ergebnis.

Methoden:

```

public double kostenVonKnoten(de.denkselbst.matrixAligner.BFSMatrixAligner.Knoten)

public java.util.ArrayList nachfolger(de.denkselbst.matrixAligner.BFSMatrixAligner.Knoten)

public de.denkselbst.matrixAligner.BFSMatrixAligner.Weg bfsalign()
public de.denkselbst.matrix.Pfad align() Alignment als Pfad liefern
public void run() run-Methode ruft nur bfsalign() auf...
public static void main(java.lang.String[ ]) ] Test / Demo: Lese Matrix aus Datei args[0] ein
und gebe Alignment auf System.out aus.

```

#### **de.denkselbst.matrixAligner.BFSMatrixAligner.Knoten**

Hierarchie:

```
java.lang.Object
```

Methoden:

```

public double kosten()
public void gibStrafe(double)

```

#### **de.denkselbst.matrixAligner.BFSMatrixAligner.Weg**

Hierarchie:

```
java.lang.Object
```

Methoden:

```

public void haengeKnotenAn(de.denkselbst.matrixAligner.BFSMatrixAligner.Knoten)
public boolean bistDuZuLangeHorizontalOderVertikalGelaufen()
public java.util.Iterator iterator()
public double kosten()
public boolean enthaeltstDuDiesenKnoten(de.denkselbst.matrixAligner.BFSMatrixAligner.Knoten)

public boolean endestDuInDiesemKnoten(de.denkselbst.matrixAligner.BFSMatrixAligner.Knoten)

public boolean endestDuIn(int, int)
public java.util.ArrayList nachfolgeWege()
public void print()

```

#### **de.denkselbst.matrixAligner.BFSMatrixAligner.Agenda**

Hierarchie:

```
java.lang.Object
```

Innere Klasse: Agenda für die Breitensuche

Methoden:

```

public boolean bistDuLeer()
public int size()
public void fuegeEinenWegEin(de.denkselbst.matrixAligner.BFSMatrixAligner.Weg)
public void fuegeWegeEin(java.util.ArrayList)
public void add(double, java.lang.Object) Objekt einfügen
public de.denkselbst.matrixAligner.BFSMatrixAligner.Weg billigster()
public void behalteNurSoviele(int)

```

#### **de.denkselbst.matrixAligner.BestCellMatrixAligner**

Hierarchie:

```

java.lang.Object
de.denkselbst.matrixAligner.MatrixAligner

```

Diese Klasse löst das folgende Problem: Im de-news Korpus tritt das Phänomen auf, dass die Reihenfolge der Nachrichtenblöcke in der englischen Datei nicht mit der deutschen Datei übereinstimmt, obwohl zu jedem Block eine Übersetzung vorhanden ist. Idee aus dem Projekt / Arno: Man nehme aus beiden Dateien nur die Überschriften und aligne diese mit einem simplen Verfahren, dass Überkreuzungen zulässt. Auf diese Weise wird ein Mapping von englischen zu deutschen Überschriften erzeugt. Anhand dieses Mappings lassen sich die Dateien in dieselbe Reihenfolge bringen. Dieser MatrixAligner: a) sucht die beste Matrixzelle, b) trägt ihre Koordinaten als Mapping ein, c) zerstört die gesamte Spalte und Zeile, in der die Zelle liegt und d) wiederholt diese Schritte, bis zu jeder Koordinate der kürzeren Matrixdimension eine entsprechende Koordinate gefunden wurde, also zu jeder Zeile (Spalte) eine Spalte (Zeile) zugeordnet wurde. Auf diese Weise wird ein 1:1 Mapping von X zu Y (also englischer zu deutscher Überschrift) erzeugt. Momentan wird das Ergebnis als Pfad geliefert. Das Ergebnis ist korrekt, führt aber zu einer unschönen Darstellung in Mavis, da die Pfadkoordinaten nicht in geeigneter Reihenfolge im Pfad stehen, um in einem Linienzug gezeichnet zu werden. Konvertiert man den Pfad in eine Koksuzuordnung, ist die Darstellung ok, die Reihenfolge jedoch unverändert. Ist die Matrix nicht quadratisch, gehen einige Spalten(Zeilen) leer aus, da auf einer 1:1 Zuordnung bestanden wird.

Methoden:

```

public de.denkselbst.matrix.Pfad align() Alignment-Methode
public void run() Die run-Methode ruft lediglich align() auf. Ergebnisse werden über das
Observer-Modell an den Interessenten geliefert. (Daher wird der Rückgabewert der Me-
thode an dieser Stelle ignoriert.)
public static void main(java.lang.String[ ] ) winziger Test: Lese Matrix aus Datei args[0]
ein und gebe sie auf System.out aus.

```

#### **de.denkselbst.matrixAligner.AStarMatrixAligner**

Hierarchie:

```

java.lang.Object
de.denkselbst.matrixAligner.MatrixAligner

```

Diese Klasse ist ein MatrixAligner, der den A\*-Suchalgorithmus für die Berechnung des Pfades einsetzt. Dreht man die Suchrichtung um (rechts unten nach links oben), so kann aufgrund der Methode, die die Nachfolgeknoten berechnet kein Ergebnis gefunden werden.

Methoden:

```

public void konfigurationsDialogAbfahren() modalen Konfigurationsdialog abfahren und Ali-
gner entsprechend konfigurieren.

```

```
public de.denkselbst.matrix.Pfad align() ermittelt (den) Alignment-Pfad  
public static void main(java.lang.String[ ] ) winziger Test: Lese Matrix aus Datei args[0]  
ein und aligne mal
```

#### **de.denkselbst.matrixAligner.AStarMatrixAligner.ConfigDialog**

Hierarchie:

```
java.lang.Object  
  java.awt.Component  
    java.awt.Container  
      java.awt.Window  
        java.awt.Dialog  
          javax.swing.JDialog
```

Innere Klasse, die innerhalb einer grafischen Oberfläche einen Konfigurationsdialog darstellt und die vom Benutzer eingestellten Werte in den Aligner einträgt.

Methoden:

```
public void actionPerformed(java.awt.event.ActionEvent)
```

### **F.5.4 Das Paket de.denkselbst.matrix**

Enthaltene Klassen:

- de.denkselbst.matrix.MatrixHistogramView
- de.denkselbst.matrix.WriteRandomMatrix
- de.denkselbst.matrix.Pfad
- de.denkselbst.matrix.Pfad.Koordinate
- de.denkselbst.matrix.Matrix
- de.denkselbst.matrix.KoksZuordnung
- de.denkselbst.matrix.KoksZuordnung.Zuordnung
- de.denkselbst.matrix.AbstandInfoDialog
- de.denkselbst.matrix.MatrixView
- de.denkselbst.matrix.MatrixView.Elem
- de.denkselbst.matrix.MatrixView.XYInfo
- de.denkselbst.matrix.MatrixView.T
- de.denkselbst.matrix.ShowAbstand
- de.denkselbst.matrix.SatzInfoDialog
- de.denkselbst.matrix.MatrixStatInfoDialog
- de.denkselbst.matrix.HTMLKocher
- de.denkselbst.matrix.ShowTags
- de.denkselbst.matrix.Convert

**de.denkselbst.matrix.MatrixHistogramView**

Hierarchie:

```

java.lang.Object
  java.awt.Component
    java.awt.Container
      javax.swing.JComponent
        javax.swing.JPanel

```

Diese Klasse ist eine grafische Komponente, die in einem eigenen Fenster ein Histogramm eines dem Konstruktor übergebenen Matrix-Objektes zeichnet.

Methoden:

**public void paint(java.awt.Graphics)** Das tatsächliche Zeichnen findet hier statt.  
**public void setzeSchrittgroesse(double)** Schrittgröße verändern  
**public void beobachtetesObjektHatSichGeregt(java.lang.Object, java.lang.String)** Beobachter: Ein update der Anzeige wird nötig.  
**public void abmelden()** Die Beobachtung des Matrix-Objektes einstellen

**de.denkselbst.matrix.WriteRandomMatrix**

Hierarchie:

```

java.lang.Object

```

Write random matrix to stdout. Values are between 0.0 and 1.0. First command line argument is taken as xdim, second as ydim. Default dims: 10 x 10 Data is written row per row from left to right. The output has the following format: xdim \t ydim \t val1 \t ... \t valn and can be read by Matrix-objects. The aim is to provide test material to Matrix-objects.

Methoden:

**public static void main(java.lang.String[ ])**

**de.denkselbst.matrix.Pfad**

Hierarchie:

```

java.lang.Object

```

Diese Klasse repräsentiert eine Liste von Koordinatenpaaren. Ein Pfad ist eine Liste von Koordinatenpaaren. Im Rahmen von Graphensuchalgorithmen ist das ein geeignetes Mittel, um einen Pfad in einer Matrix zu beschreiben. Längere horizontale Abschnitte in einem Pfad werden als n:m - Alignments aufgefasst, durch eine Konvertierung des Pfades in eine KoksZuordnung wird dem Rechnung getragen. MatrixAligner liefern ihre Ergebnisse derzeit als Pfade. Pfade wissen, wie sie sich in KoksZuordnungen konvertieren können. Die Klasse KoksZuordnung bietet eine Konstruktor, der einen Pfad entgegen nimmt. Die KoksZuordnung ruft dann die Methode toWagnerString beim Pfad auf und erhält so einen String, den sie interpretieren kann. Quelle und Info müssen aus einer Matrix übernommen werden, deshalb hat die toWagnerString Methode Argumente, in denen diese Strings übergeben werden müssen.

Methoden:

**public void haengeAn(int, int)** eine Koordinate ans Ende des Pfades anhängen  
**public void haengeVorneAn(int, int)** eine Koordinate an den Anfang des Pfades anhängen  
**public java.util.Iterator iterator()** einen Iterator über die Elemente des Pfades liefern

```

public void umdrehen() Pfade umdrehen
public java.lang.String toString() Einfache Ausgabe
public java.lang.String toWagnerString(java.lang.String, java.lang.String) Pfad als String
im Wagner-Format liefern. Verwendungszweck ist sowohl textuelle Ausgabe, als auch
Schritt bei der Konvertierung in KoksZuordnung. Die Strings quelle und info müssen
übergeben werden, Matrix- Objekte verfügen über entsprechende Strings, von dort soll-
ten diese Informationen also beschafft werden. Der Algorithmus verfolgt den Pfad von
Anfang bis Ende und verwandelt ihn in eine Folge von Segment-Zuordnungen. Eine Dia-
gonalbewegung nach rechts unten schliesst die aktuelle Segmentzuordnung ab und be-
ginnt eine neue. Durch vertikalen oder horizontalen Pfadverlauf verknüpfte Sätze werden
an die aktuelle Segment-Zuordnung angehängt.
public static void main(java.lang.String[ ] ) Demonstration / Test

```

#### **de.denkselbst.matrix.Pfad.Koordinate**

Hierarchie:

```
java.lang.Object
```

Kapselt x- und y-Koordinate

Es sind keine Methoden enthalten.

#### **de.denkselbst.matrix.Matrix**

Hierarchie:

```
java.lang.Object
```

Diese Klasse repräsentiert eine aus double Werten bestehende Abstandsmatrix. Die Abstandsmatrix als solche ist die Schnittstelle zwischen Abstandsmaß und Aligner. Daher ist diese Klasse ein sehr wichtiger und zentraler Bestandteil des Aligners. Neben den Abstandswerten enthalten Objekte dieser Klasse zwei Strings, die den Pfad zur Ausgangsdatei und Zusatzinformationen enthalten (quelle, info). Im Allgemeinen werden Objekte dieser Klasse über einen Konstruktor mit Dateinamen aus einer Datei, bzw. aus STDIN mit Werten gefüllt. Das benutzte Dateiformat hat folgende Form: Quelle-String \n Info-String \n Breite Hoehe \n Wert1 \n Wert2 \n ... WertN Diese Klasse erkennt neben \n auch anderen Whitespace als Trenner zwischen den Abstandswerten an. (Siehe entsprechende Methode.) Die Abstandswerte, so wurde es im Projekt vereinbart, liegen im Intervall von 0 bis 1. Damit die Heuristik des A\*-Aligners ihre Funktionalität erfüllen kann, müssen alle Abstandswerte  $\geq 0$  sein. Die Methode normalisiereDich() verschiebt die Abstandswerte einer Matrix in das Intervall [0.1 ... 1]. Dies erhöht auch den Kontrast in einer Matrix aus TG-Werten, die in einem sehr kleinen Intervall liegen. Eine Reihe von Methoden mit zugegebenermaßen recht ausschweifigen Namen erlauben den Zugriff auf die Verkapselten Daten. Am wichtigsten sind holeDatumVonStelle(int x, int y) und wasSindVonHierDieUnterschaetztenKostenZurRechtenUnterenEcke(). In dem GUI-Tool Mavis gibt es viele Objekte, die sich für Änderungen (Transformation der Daten, neue Datei geladen) interessieren, daher wurde ein einfaches Observer-Modell geschaffen (package de.denkselbst.beobachtung), das vom Matrix-Objekt unterstützt wird. Sonstiges: Die Zugriffsmethoden sind derzeit nicht synchronisiert.

Methoden:

```

public java.lang.String holeQuelle() liefert den Quellen-String
public java.lang.String holeInfo() liefert den Info-String
public java.lang.String holeDateiname() liefert den Dateinamen als String
public int holeBreite() liefert die Breite der Matrix
public int holeHoehe() liefert die Hoehe der Matrix

```

**public double** [ ] **kopieDerDaten()** Kopie von `daten[][]` liefern. Eine solche Kopie der `daten[][]` kann von anderen Objekten ohne Auswirkungen auf andere Programmteile destruktiv verändert werden.

**public double** **holeDatumVonStelle(int, int)** liefert den Wert der Matrix an der Stelle (x,y)

**public void** **setzeDatumAnStelle(int, int, double)** setzt den Wert der Matrix an der Stelle (x,y) auf `datum` Minimum und Maximum werden gepflegt

**protected void** **geaendert()** intern aufrufen, um Zustandsänderung klarzumachen altes Minimum und Maximum verlieren Gültigkeit z.B. nach Transformation und Stutzen!

**public double** **wasIstDerKleinsteWert()** Kleinsten Wert (Minimum) der Matrix liefern

**public int** **getMinX()** x-Koordinate des Minimums liefern

**public int** **getMinY()** y-Koordinate des Minimums liefern

**public double** **wasIstDerGrosessteWert()** Maximum der Matrix liefern

**public double** **wasSindVonHierDieUnterschaetztenKostenZurRechtenUnterenEcke(int, int)** Unterschätzte Kosten zur rechten unteren Ecke liefern (Diagonale Distanz)

**public double** **minKost(int, int, int, int)** Mindestkosten von x1,y1 nach x2,y2 liefern

**public int** **wievieleSchritteVonHierZurRechtenUnterenEcke(int, int)** minimale Anzahl Schritte von xPos, yPos zum Ziel (rechte untere Ecke)

**public void** **stutzenAuf(int, int)** Matrix auf die gegebenen Dimensionen zurechtstutzen

**public void** **fillColumn(int, double)** Eine komplette Spalte der Matrix auf Wert w setzen

**public void** **fillRow(int, double)** Eine komplette Zeile der Matrix auf Wert w setzen

**public double** **berechneMittelwert()** Mittelwert berechnen

**public double** **berechneVarianz()** Varianz berechnen

**public double** **berechneStandardabweichung()** Standardabweichung berechnen

**public void** **normalisiereDich()** Transformation: Daten normalisieren auf Intervall [0.1 .. 1]

**public void** **rangfolgenClustering(int)** Daten transformieren: Clusterbildung nach Rangfolge der Daten

**public void** **setzeDatenAusDatei(java.lang.String)** Matrix mit neuen Daten aus Datei füttern

**public void** **setzeDatenAusStdin()** Matrix mit neuen Daten aus Stdin füttern

**public void** **setzeDatenAusGepuffertemLeser(java.io.BufferedReader)** Matrix mit neuen Daten aus `BufferedReader` füttern

**public java.lang.String** **toString()** Matrix als String liefern (billige Ausgabe zu Diagnosezwecken)

**public void** **beobachterAnmelden(de.denkselbst.beobachtung.Beobachter)** Einen neuen Beobachter in die Liste der im Falle einer Änderung zu benachrichtigen Objekte aufnehmen.

**public void** **beobachterAbmelden(de.denkselbst.beobachtung.Beobachter)** einen Beobachter aus der Liste der zu benachrichtigen Objekte entfernen.

**public void** **beobachterBenachrichtigen(java.lang.String)** alle Beobachter ob einer Änderung benachrichtigen, Referenz auf das Beobachtete Objekt (`this`) und Nachricht String werden mitgeliefert

**public static void** **main(java.lang.String[ ] )** winziger Test: Lese Matrix aus Datei `args[0]` ein und gebe sie auf `System.out` aus.

### de.denkselbst.matrix.KoksZuordnung

Hierarchie:

`java.lang.Object`

Diese Klasse repräsentiert ein Alignment. Ein Alignment besteht aus Zuordnungen (KoksZuordnung,Zuordnung), die n Sätze (Segmente) aus L1 mit m Sätzen aus L2 verbinden. Eine Liste solcher Zuordnungen, die sequentiell die Menge aller Sätze bedeckt, ist eine KoksZuordnung. KoksZuordnungen können im Wagner-Format textuell ausgegeben werden und stellen damit die Schnittstelle vom Aligner zu den nachfolgenden Modulen dar.

Methoden:

```

public java.lang.String holeInfo() Info-String liefern
public java.lang.String holeQuelle() Quelle liefern
public java.lang.String toString() Erzeugt Wagner-Format (Datei, nicht Pizza)
public java.util.ArrayList betroffeneKoordinaten() Liefert die Liste der Koordinaten, die
    im Alignment liegen x,y,x1,y1,x2,y2,...xn,yn
public void printKoord() Zum Test mal die betroffenen Koordinaten ausgeben
public java.util.ArrayList getZuordnungen() Zuordnungen rausrücken (Wird benutzt um
    HTML-Tabelle in HTMLKocher zu bauen...)
public static void main(java.lang.String[ ] ) winziger Test: Lese Matrix aus Datei args[0]
    ein und gebe sie auf System.out aus.

```

#### de.denkselbst.matrix.KoksZuordnung.Zuordnung

Hierarchie:

```
java.lang.Object
```

Methoden:

```

public void addL1(int) Füge Liste 1 eine (Satz)Nummer hinzu
public void addL2(int) Füge Liste 2 eine (Satz)Nummer hinzu
public java.util.ArrayList getL1() Liefere Liste1 nach aussen (z.B. wird damit an anderer
    Stelle eine HTML-Tabelle mit den tatsächlichen Sätzen erzeugt)
public java.util.ArrayList getL2() Liefere Liste2 nach aussen (z.B. wird damit an anderer
    Stelle eine HTML-Tabelle mit den tatsächlichen Sätzen erzeugt)
public java.lang.String toString() Gibt die Zuordnung im Wagnerformat aus
public void betroffeneKoordinaten(java.util.ArrayList) Trägt in der Liste koord die ent-
    sprechenden Koordinaten der Zuordnung ein x1,y1,x2,y2,...xn,yn

```

#### de.denkselbst.matrix.AbstandInfoDialog

Hierarchie:

```

java.lang.Object
  java.awt.Component
    java.awt.Container
      java.awt.Window
        java.awt.Dialog
          javax.swing.JDialog

```

Nach dem Vorbild von SatzInfoDialog geklonte Klasse zum Anzeigen von Debugging-Informationen für Abstandsmaße. Autor: Joachim.

Methoden:

```

public de.denkselbst.matrix.ShowAbstand getShowAbstandObject()

```

```

public void updateQuelle()
public void show(int, int)
public void dead()
public void setzeMatrix(de.denkselbst.matrix.Matrix) neues Matrix Objekt einpflanzen
public void beobachtetesObjektHatSichGeregt(java.lang.Object, java.lang.String)
public static void main(java.lang.String[ ] )

```

### de.denkselbst.matrix.MatrixView

Hierarchie:

```

java.lang.Object
  java.awt.Component
    java.awt.Container
      javax.swing.JComponent
        javax.swing.JPanel

```

Diese Klasse stellt eine Matrix grafisch dar und stellt Fenster für die Alignmentpfade bereit. Fast alles sichtbare in Mavis stammt von hier.

Methoden:

```

public void setzeMatrix(de.denkselbst.matrix.Matrix) neues Matrix Objekt einpflanzen
public void fuegeAlignmentPfadHinzu(de.denkselbst.matrix.Pfad, java.lang.String) einen
    Pfad in die Liste der Alignment-Pfade aufnehmen, Info-String ist mitzuliefern!
public void entferneSelektiertenAlignmentPfad() den selektierten Pfad aus der Liste der
    Alignment-Pfade löschen
public void zeigeSelektiertenAlignmentPfadAlsHTML() den selektierten Pfad aus der Li-
    ste der Alignment-Pfade als HTML anzeigen
public boolean istEtwasSelektiert()
public void loescheAllePfade() Liste der Alignment-Pfade komplett löschen
public void speichereSelektiertenAlignmentPfad(java.io.File) Schreibt den selektierten Pfad
    (so vorhanden) in die Datei file
public void fuegeKoksZuordnungEin(de.denkselbst.matrix.KoksZuordnung, java.lang.String)
    KoksZuordnung in die Liste der Pfade aufnehmen
public void paint(java.awt.Graphics) In dieser Methode finden das Zeichnen der Matrix
    und der Pfade statt.
public java.awt.Color getColor(double) einen double Wert zwischen 0 und 1 gegen einen
    Grünton eintauschen: 0 = sehr helles Grün, 1 = sehr dunkles Grün, man könnte auch
    Schwarz dazu sagen
public void setzeBlockGroesse(int, int) Blockgröße setzen (Zeichnen)
public void zeigeDiagonale(boolean) Schalter: Diagonale zeichnen oder nicht? (Zeichnen)
public void beobachtetesObjektHatSichGeregt(java.lang.Object, java.lang.String)
public void mouseDragged(java.awt.event.MouseEvent) MouseMotionListener und Mou-
    seListener
public void mouseMoved(java.awt.event.MouseEvent)
public void mouseClicked(java.awt.event.MouseEvent)
public void mouseEntered(java.awt.event.MouseEvent)
public void mouseExited(java.awt.event.MouseEvent)
public void mousePressed(java.awt.event.MouseEvent)
public void mouseReleased(java.awt.event.MouseEvent)

```

**de.denkselbst.matrix.MatrixView.Elem**

Hierarchie:

```
java.lang.Object
```

Innere Klasse: Bündelt Pfad/KoksZuordnung mit Farbe, Info und Sichtbarkeitsstatus

Es sind keine Methoden enthalten.

**de.denkselbst.matrix.MatrixView.XYInfo**

Hierarchie:

```
java.lang.Object
  java.awt.Component
    java.awt.Container
      java.awt.Window
        java.awt.Dialog
          javax.swing.JDialog
```

Memberklasse, die in einem eigenen Fenster die aktuellen MatrixKoordinaten und den zugehörigen Wert anzeigt

Methoden:

**public void update(int, int, java.awt.Color, double)** Die neuen Werte zur Anzeige bringen

**public void dead()** Teilt diesem Objekt mit, dass keine anzeigbaren Werte vorliegen. z.B. Maus ausserhalb des Fensters

**de.denkselbst.matrix.MatrixView.T**

Hierarchie:

```
java.lang.Object
  javax.swing.table.AbstractTableModel
```

eklige Memberklasse, die als Tablemodel dient

Methoden:

**public void updateTable()**

**public int getColumnCount()**

**public int getRowCount()**

**public java.lang.String getColumnName(int)**

**public java.lang.Class getColumnClass(int)**

**public java.lang.Object getValueAt(int, int)**

**public boolean isCellEditable(int, int)**

**public void setValueAt(java.lang.Object, int, int)**

**de.denkselbst.matrix.ShowAbstand**

Hierarchie:

```
java.lang.Object
```

Nach dem Vorbild von ShowTags geklonte Klasse zum Anzeigen von Debugging-Informationen für Abstandsmaße. Autor: Joachim.

Methoden:

```
public java.lang.String getAbstandinfo(int, int)
public int xmax()
public int ymax()
public static void main(java.lang.String[ ]) ] kleines Main
```

**de.denkselbst.matrix.SatzInfoDialog**

Hierarchie:

```
java.lang.Object
  java.awt.Component
    java.awt.Container
      java.awt.Window
        java.awt.Dialog
          javax.swing.JDialog
```

Diese Klasse zeigt in einem Fenster Sätze textuell an, die zu der aktuell selektierten Matrixzelle gehören. Dazu wird dem Konstruktor ein Matrix-Objekt übergeben. SatzInfoDialog meldet sich bei der Matrix als Beobachter an und besorgt sich die Quell- Information derselben. Es wird ein ShowTags-Objekt angelegt, das mittels der Quelleninformation möglicherweise die entsprechenden Textdateien lesen kann. Das MatrixView-Objekt reagiert auf Mausbewegungen und weist sein SatzInfoDialog-Objekt an, die zur aktuell unter dem Mauscursor befindlichen Matrixzelle gehörenden Sätze anzuzeigen.

Methoden:

```
public de.denkselbst.matrix.ShowTags getShowTagsObject() Liefert benutztes ShowTags-Objekt
public void updateQuelle() Erzeugt ShowTags-Objekt für internen Gebrauch
public void show(int, int) Sorgt dafür, das die Sätze x und y textuell im Fenster angezeigt werden
public void dead() Wird aufgerufen, wenn der Mauszeiger nicht über einer Matrixzelle steht. In diesem Fall kann nichts angezeigt werden.
public void setzeMatrix(de.denkselbst.matrix.Matrix) neues Matrix Objekt einpflanzen
public void beobachtetesObjektHatSichGeregt(java.lang.Object, java.lang.String)
```

**de.denkselbst.matrix.MatrixStatInfoDialog**

Hierarchie:

```
java.lang.Object
  java.awt.Component
    java.awt.Container
      java.awt.Window
        java.awt.Dialog
          javax.swing.JDialog
```

Diese Klasse zeigt in einem eigenen Fenster ?Dateiname, Mittelwert, Varianz, Standardabweichung der dem Konstruktor übergebenen Matrix an.

Methoden:

```
public void setzeMatrix(de.denkselbst.matrix.Matrix) neues Matrix Objekt einpflanzen
public void beobachtetesObjektHatSichGeregt(java.lang.Object, java.lang.String)
```

#### **de.denkselbst.matrix.HTMLKocher**

Hierarchie:

```
java.lang.Object
```

Diese Klasse enthält eine statische Methode, die KoksZuordnung unter Zuhilfenahme eines ShowTags-Objektes und diversen info-Strings ls HTML-String liefert. So kann ein Alignment in einer HTML-View textuell dargestellt werden.

Methoden:

```
public static java.lang.String kocheTabelle(de.denkselbst.matrix.KoksZuordnung, de.denkselbst.matrix.ShowT...
    Diese statische Methode gibt eine KoksZuordnung @param k unter Zuhilfenahme eines
    ShowTags-Objektes @param s und diversen info-Strings als HTML-String zurück.
```

#### **de.denkselbst.matrix.ShowTags**

Hierarchie:

```
java.lang.Object
```

Diese Klasse besorgt, falls möglich, die zu einer Matrix gehörenden Texte und liefert auf Anfrage Sätze (Strings) zu übergebenen Satznummern (int).

Methoden:

```
public java.lang.String getX(int) Liefert Satz x aus L1 als String.
public java.lang.String getY(int) Liefert Satz y aus L2 als String.
public int xmax()
public int ymax()
public static void main(java.lang.String[ ]) Kleines Main zu Test- / Demozwecken
```

#### **de.denkselbst.matrix.Convert**

Hierarchie:

```
java.lang.Object
```

Die Main-Methode dieser Klasse veranlasst, das eine zeilenweise eingelesene Matrix spaltenweise ausgegeben wird. inputdatei = argument[n], outputdateiname = inputdateiname+„converted“ Da es nach einigen Anlaufschwierigkeiten mit dem Matrixformat keine Probleme mehr gab, war es schon früh nicht mehr nötig, Matrizen mit diesem Programm zu drehen.

!!Diese Klasse ist daher obsolet!! Allerdings ist sie noch immer Teil des CVS-Baumes. Der Code zum Einlesen einer Matrix wurde hier dupliziert, was eine Sünde ist.

Methoden:

```
public static void main(java.lang.String[ ]) ]
```

## F.5.5 Das Paket de.denkselbst.mavis

Enthaltene Klassen:

- de.denkselbst.mavis.Mavis
- de.denkselbst.mavis.ShowHTMLFrame
- de.denkselbst.mavis.MACV
- de.denkselbst.mavis.StutzenDialog

### de.denkselbst.mavis.Mavis

Hierarchie:

```
java.lang.Object
  java.awt.Component
    java.awt.Container
      java.awt.Window
        java.awt.Frame
          javax.swing.JFrame
```

Diese Klasse setzt die grafische Oberfläche von Mavis zusammen und kümmert sich um Teile des Event-Handlings. Der Code ist immer mal wieder auf die Schnelle geändert worden. Er ist nicht besonders schön und auch mit der heißen Nadel gestrickt. Aber das muss so sein, schließlich handelt es sich um einen Prototypen.

Methoden:

```
public void actionPerformed(java.awt.event.ActionEvent)
```

```
public void itemStateChanged(java.awt.event.ItemEvent)
```

```
public static void main(java.lang.String[ ] ) Main-Methode bringt den Stein ins Rollen.
```

### de.denkselbst.mavis.ShowHTMLFrame

Hierarchie:

```
java.lang.Object
  java.awt.Component
    java.awt.Container
      java.awt.Window
        java.awt.Frame
          javax.swing.JFrame
```

Diese Klasse öffnet ein HTML-Fenster auf der Oberfläche, das per Konstruktor mit HTML-Code gefüllt wird. Zweck ist das Anzeigen eine Alignments in Textform. Das Fenster ist nach seiner Erzeugung unabhängig von anderen Programmteilen und wird von keinem Objekt kontrolliert. Es reagiert lediglich auf das Schließen-Event, das vom Schließen-Knopf im Fensterrahmen gesendet wird.

Methoden:

```
public static void main(java.lang.String[ ] ) Kleiner Test
```

**de.denkselbst.mavis.MACV**

Hierarchie:

```
java.lang.Object
  java.awt.Component
    java.awt.Container
      java.awt.Window
        java.awt.Frame
          javax.swing.JFrame
```

Objekte dieser Klasse rufen einen MatrixAligner ins Leben und observieren ihn. Die Nachrichten des Aligners werden in einem eigenen Fenster dargestellt, das von dieser Klasse erzeugt wird. Objekte dieser Klasse werden von Mavis.java ins Leben gerufen, wenn der Benutzer ein Alignment veranlasst.

Methoden:

```
public void actionPerformed(java.awt.event.ActionEvent) ActionListener
public void alignmentFertig(de.denkselbst.matrix.Pfad, boolean) AlignmentObserver
public void statusInformation(java.lang.String, boolean) AlignmentObserver
```

**de.denkselbst.mavis.StutzenDialog**

Hierarchie:

```
java.lang.Object
  java.awt.Component
    java.awt.Container
      java.awt.Window
        java.awt.Dialog
          javax.swing.JDialog
```

Definiert einen Dialog, mit dem eine Matrix verkleinert werden kann. Wird von Mavis.java aus ins Leben gerufen.

Methoden:

```
public void actionPerformed(java.awt.event.ActionEvent)
public void stateChanged(javax.swing.event.ChangeEvent)
public int getXVal()
public int getYVal()
public boolean cancelled()
public static void main(java.lang.String[ ]) Kleiner Test / Demo
```

**F.5.6 Das Paket de.denkselbst.kommandozeile**

Enthaltene Klassen:

- de.denkselbst.kommandozeile.HeadingMapper4Arno
- de.denkselbst.kommandozeile.Align

**de.denkselbst.kommandozeile.HeadingMapper4Arno**

Hierarchie:

`java.lang.Object`

Die Main-Methode dieser Klasse liest Matrizen von Stdin und schreibt 1:1 Zuordnungen im Wagner-Format nach Stdout. Sonstige Meldungen werden auf Stderr ausgegeben. Idee und Algorithmus sind in der Dokumentation der Klasse `de.denkselbst.matrixAligner.BestCellMatrixAligner` beschrieben.

Methoden:

```
public static void main(java.lang.String[ ] )
```

**de.denkselbst.kommandozeile.Align**

Hierarchie:

`java.lang.Object`

Die Main-Methode dieser Klasse liest Matrizen von STDIN und schreibt Alignments nach STDOUT. Die wichtigsten Schritte sind es, den Eingabestrom zu besorgen, ein Matrixobjekt und ein damit verbundenes MatrixAlignerobjekt zu erstellen, die Matrix aus STDIN zu füllen, zu normalisieren, den MatrixAligner das Alignment (als Pfad) finden lassen, den Pfad in eine KoksZuordnung umzuwandeln und das Ergebnis auf STDOUT auszugeben.

Da es offensichtliche Performanzvorteile gibt, wenn nicht für jede Matrix einmal die JAVA-VM gestartet werden muss, können `align` mehrere Aufträge pro Aufruf erteilt werden. Dazu muss der erste Parameter ein Integer-Wert sein, der die Anzahl der Aufträge enthält. Um den obigen Code-Ausschnitt herum existiert eine Schleife. Wird `align` ohne Angabe der Auftragsanzahl aufgerufen, wird diese Schleife nur einmal durchlaufen.

Methoden:

```
public static void main(java.lang.String[ ] )
```

## Anhang G

# Phrasenalignment – Dokumentation

## G.1 Tools und Optionen

### G.1.1 `getsegs.py`

Das Python-Skript `getsegs.py` sucht alle Segmente aus dem Korpus heraus, die einen Satz einer bestimmten Länge enthalten. Man kann übergeben, ob die Sätze aus dem Korpus oder dem Wörterbuch stammen sollen, und wieviele Sätze es sein sollen. Die Ids der Sätze werden in eine Datei mit der Endung `.segs` geschrieben, die Sätze selbst in eine Datei mit der Endung `.txt`, damit der Inhalt und z. B. das Alignment überprüft werden können. Den Namen der Datei vor der Endung muss man auch angeben. Die Reihenfolge der Parameter ist `getsegs.py <segmentgroesse> <segmentanzahl> wb|ko <filename>`. Möchte man also 100 Segmente der Länge 7 aus dem Korpus in die Datei `ph7` schreiben, wäre die Syntax `getsegs.py 7 100 ko ph7`.

### G.1.2 `zuordnung2.py`

Dieses Programm wird nicht aufgerufen, es ist nur eine Library, die den im Kapitel 8 auf Seite 76 beschriebenen Algorithmus implementiert.

### G.1.3 `phrasealign.py`

Diese Skript bekommt den Namen einer Datei übergeben, die die Ids von Segmenten in der Datenbank enthält, die man alignen möchte. Wurde das Segment schon aligniert, kommt eine Meldung, und es wird übersprungen. Sollen die oben gefundene Segmente der Länge 7 aligniert werden, so ist der Aufruf `phrasealign.py ph7.segs`.

Die Ausgabe besteht aus den Sätzen, den Wörtern, die als übersetzt gekennzeichnet wurden, und den gefundenen Phrasen. Außerdem werden die zugeordneten Phrasen in die Datenbank eingetragen, in die Tabellen `phrase_example` und `phrase_count`.

## G.2 Dateiformate

In den `.segs`-Dateien stehen nur die Ids der Segmente, durch Newlines getrennt. Die `.txt`-Dateien sind etwas komplizierter aufgebaut. Zunächst wird die Id des Segments ausgegeben, dann die Wörter des deutschen Segments mit ihren Tags. Es folgt eine Reihe von Sternchen, dann die englischen Wörter + Tags. Die Segmente sind durch eine Reihe von Strichen getrennt.

`phrasealign.py` gibt zunächst die Sätze des zu alignierenden Segments aus. Dann kommt eine Zeile mit `*-*`, danach werden die einander zugeordneten Phrasen ausgegeben und die Tags, die sie hatten. Abgetrennt werden die einzelnen Phrasen durch eine Reihe von `#`.

Beispiel:

Die Kommission beschließt ein Mehrjahresprogramm zur Förderung erneuerbarer Energieträger.
--



```
#####  
['promotion'] -> ['Förderung']  
['promotion'] -> ['Förderung']  
['renewable'] -> ['<unknown>']  
['renewable'] -> ['<unknown>']  
['energy'] -> ['<unknown>']  
['energy'] -> ['<unknown>']  
['energy'] -> ['Energie']  
['energy'] -> ['Energie']  
['Förderung'] -> ['<unknown>']  
['Förderung'] -> ['<unknown>']  
['Förderung'] -> ['promotion']  
['Förderung'] -> ['promotion']
```

### G.3 Schnittstellen

Die Methode `ordneZu` des Skripts `zuordnung2.py` erwartet 2 Segmente, und ordnet diese dann zu. Sie liefert eine Liste von Tupel, die eine deutsche und eine englische Phrasen enthalten, die einander zugeordnet werden können.

# Anhang H

## Datenbank – Dokumentation

### H.1 SQL-Skripte zum Erstellen der Datenbank

Es gibt einige SQL-Skripte, die verschiedene Aufgaben erfüllen:

- Anlegen der Datenbank: `db.sql`, siehe Kapitel O.1.1,
- Löschen der Datenbank: `mrprper.sql` siehe Kapitel O.1.4,
- Leeren einiger Tabellen: `clean.sql`, siehe Kapitel O.1.3
- Erzeugen von Indizes: `index.sql`, siehe Kapitel O.1.2

Benutzt man `mysql`, und hat man eine Datenbank erzeugt, so kann man die Tabellen anlegen mit `mysql -u <user> -p <datenbank> < db.sql`. Die Benutzung der anderen Skripte ist analog.

### H.2 Python-Datenbank-Schnittstelle

#### H.2.1 RegionVonDB

Abstammend von `Region` (Kapitel F.5.1) und analog zu `RegionVonDatei` (Kapitel F.4.1) repräsentieren diese Klassen eine `Region`, die aus der Datenbank geladen wird. Der Konstruktor bekommt eine Liste von `Ids` übergeben, aus denen sich diese `Region` zusammensetzt (`Segment`: Liste von Sätzen, `Satz`: Liste von Wörtern). Erst, wenn direkt auf ein solches Element zugegriffen wird, erfolgt der Zugriff auf die Datenbank. Mithilfe der Klasse `RegionenMit` (s. Kapitel F.5.1) greift man auf die Unterregionen zu.

#### H.2.2 Wörterbücher

##### `wb.py`

Das Modul `wb` enthält u.a. die Klassen `WoerterbuchVonDB` und `LemmatabuchVonDB`, deren Instanzen einen einfachen Zugriff auf die Wörterbucheinträge und erkannten Phrasen in der `KoKS`-Datenbank erlauben. Sie können wie normale Python-Dictionaries verwendet werden.

```
koks@nunix:~/align > python
Python 1.5.2 (#1, Sep 20 2000, 19:42:01) [GCC 2.95.2 19991024 (release)] on linux2
Copyright 1991-1995 Stichting Mathematisch Centrum, Amsterdam
>>> from wb import *
>>> from DatabaseAPI import regiondb, db
>>> dbconn = db.Db()
>>> regiondb.setDefaultDbconn(dbconn)
>>> lembuch = LemmatabuchVonDB('de', 'en', 'erkannt')
Benutze Server ('nunix.cl-ki.uni-osnabrueck.de', 29295)
>>> result = lembuch[['ins', 'Leben', 'rufen']]
```

```
>>> print result
[(-1, <662123 SegmentVonDb 1 dbconn=135638496>, [<662123 SegmentVonDb 2 dbconn=135638496>])]
>>> print result[0][1].getText(), '-', result[0][2][0].getText()
ins Leben gerufen - was launched
>>> dbconn.close()
```

### filterwb.py

Mithilfe dieses Moduls kann man einen Satz a) in Unterphrasen zerlegen, und b) nur die Unterphrasen mit relevanten Tags herausuchen. Parallel zu den Tagfolgen (s. Kapitel 8.2) wurden die einzelnen Tags bewertet, um so Wörter aus geschlossenen Wortklassen niedriger zu bewerten als Nomen, Verben etc.

Die abstrakte Klasse `Relevant` hat zwei Methoden, `relevant()`, die von allen Unterklassen implementiert werden sollte, die relevante Tags bewerten (wie `RelevantNachTagP` und `RelevantNachTagN`, und `p_getSlices()`, mit dem man den Satz in Unterphrasen zerlegen kann, wie es z. B. `RelevantMax` oder `RelevantInseln` machen.

Die Klasse `WoerterbuchFilter` bekommt ein solches `Relevant`-Objekt übergeben, und liefert dann nur solche Übersetzungen zurück, die von dem Filter als relevant gekennzeichnet wurden. Es kann genau wie `WoerterbuchVonDb`, das nicht filtert, wie ein Python-Dictionary benutzt werden.

## H.3 Abfrage-Schnittstelle

Da die Datenbankzugriffe oft langsam sind, haben wir selbst einen Server zum Zugriff auf den getaggten Korpus und auf die Phrasenerkennung entwickelt, der Anfragen `cached`, spezielle Anfragen ermöglicht und auch als Backend für den Web-Zugriff (s. Kapitel 10.1) dient.

### H.3.1 querydb.py

Das Modul ist zur Abfrage von Daten aus der Datenbank gedacht. Zusätzlich stellt es Klassen zur Verfügung, um die Schnittstelle als Server bereitzustellen. Das Skript `align/demo-qdb` startet einen solchen Server (Option `-s`) oder ermöglicht den interaktiven Zugriff auf einen Server.

Da der Server weltweit erreichbar ist, erfolgt der volle Zugriff auf die Datenbank nur nach erfolgreicher Authentifikation, siehe auch `align/DatabaseAPI/secret.py`. Das Programm `demo-qdb` gestartet mit der Option `-c` und das Python-Modul `align/DatabaseAPI/qclient.py` erledigen dies automatisch.

Im Servermodus muss der Client seine Fragen in einem Stück und newlineterminiert zum Server schicken. Der Server sendet dagegen die (eventuell mehrzeilige) Antwort stückweise und schliesst sie mit `<DONE>` + Newline ab. Fehlermeldungen erkennt man daran, dass sie mit `<ERROR>` + Newline beginnen. Eine Ausnahme ist der Authentifizierungsdialog.

### Korpus auswählen

Mit `u:<Liste von IDS>` kann man die Korpora wählen, die für die Satz- und Segmentabfragen benutzt werden (für Wörter und Grundformen ist dies schwierig...). Die einzelnen Elemente müssen mit Kommata getrennt sein.

Beispiel:

```
u:(1,2,3)
u:wb
u:erkannt
u:wb+erkannt
u:good
u:all
```

In `config.py` ist die Stringkonstante `wbids` für die WB-Ids vordefiniert. I.d.R. sollte man aber wie im Beispiel `wb`, `good` usw. verwenden, da die `config`-Werte falsch sind, wenn Client und Server für verschiedene Datenbanken konfiguriert sind.

## Datenbank abfragen

Die Syntax der Batchabfragen ist wie folgt:

Mit `t:<typ>` wählt man den Typ der Abfrage aus. Zur Zeit werden `t:grundform`, `t:wort`, `t:segment`, `t:satz` und `t:text` unterstützt.

Mit `s:<info>` wählt man die Informationen aus, die man bekommen möchte. `s:all` wählt alles verfügbare aus. `s:text` gibt IMS-Tagzeilen aus. Die einzelnen Regionen der Antwortliste werden durch Delimiter getrennt, z.B. `<segmentgrenze>` für `t:segment`. `s:id` gibt die IDs der Elemente aus. `s:count` gibt die Anzahl der Elemente des Ergebnisses auf oberster Ebene zurück. Mit `s:r` erhält man eine Python-Repräsentation des Ergebnisses, mit `s_lr` werden Listen und Regionen ausgepackt, ansonsten wie `s:r`. `s:_s` liefert eine String-Repräsentation des Ergebnisses, `s:_ls` packt Listen und Regionen aus, ansonsten wie `s:_s`. `s:_r80` gibt die Länge der `s:r` Ausgabe, gefolgt von maximal so vielen Zeichen der `s:r` Ausgabe, dass 80 Zeichen nicht überschritten werden, `s:_lr80` analog zu `s:_r80` und `s:_lr`.

`w:<einschränkung>` schließlich gibt dem Programm die Information, die es braucht, um Infos zu liefern.

Die einzelnen Segmente sind durch Semikolon getrennt: `t:<typ>;s:<info>;w:<einschränkung>`  
 Warnung: Derzeit bewirkt der Parameter `w:lang` keine Einschränkung der Ergebnismenge, sondern entscheidet, für welche Sprachen Ergebnisobjekte erstellt werden sollen, oder welcher Tagger aufzurufen ist.

**grundform:** Man muss immer die Grundform angeben, nach der gesucht wird, und zwar mit `w:name=<grundform>`.

Beispiel:

```
t:grundform;s:all;w:name=haben
```

**wort:** Man kann zu einem Wort die Grundform und das Tag ausgeben. Auch hier muss mit `w:name=<wort>` das gesuchte Wort angegeben werden. Beispiel:

```
t:wort;s:grundform;w:name=hat
```

**segment:** Segment können anhand ihrer Id und anhand eines Wortes abgefragt werden. Als Information bekommt man die Sätze, die zu diesem Segment gehören. Die Infos gibt man mit `w:id=<id>` oder `w:wort=<wort>` an. Auch kann man die Sprache des Segments auswählen, mit `w:lang=deutsch,englisch`. Analog zum Satz können `match=<macth>`, `lemmata=<lemmata>` und `tags=<tags>` verwendet werden. (s.u.)

Mit `strict=true` kann Reihenfolge und Kontinuität verlangt werden. Des weiteren gibt es `match_l1` und `match_lf`, die nach Lemmata matchen, bzw. sogar mit einem schnelleren Algorithmus. Mit `phrases=<phrasen>` wird die Eingabe nach Phrasen durchsucht. Dazu ist die Angabe einer Sprache erforderlich. Mit `_fs=<wert>` kann ein Schwellwert des Wörterbuchfilters eingestellt werden.

`bid=<id>` bzw. `beispiel` zum `segment` mit `id=<id>` liefert Segmente, die als Beispiel für die angegebene Phrase eingetragen sind, oder die leere Liste, wenn die `id` nicht für eine vom Phrasenerkennner eingetragene Phrase steht.

Mit `max=` kann die Anzahl der Segmente beschränkt werden.

Beispiel:

```
t:segment;s:all;w:wort=haben
t:segment;s:ids;w:id=16,lang=englisch
t:segment;s:text;w:phrases=Ich habe <i> nichts </i> zu sagen.,lang=de
t:segment;s:text;w:bid=441676
```

**satz:** Sätze können auch aufgrund ihrer Id und eines Wortes abgefragt werden. Als Information bekommt man alle Wörter eines Satzes. Gibt man `match=<Phrase>` an, so werden nur die Sätze gesucht, die genau die Wörter der angegebenen Phrase beinhalten. Bei `lemmata=<wort1 wort2...>` werden alle Sätze gesucht, die alle angegebenen Grundformen beinhalten.

Beispiel:

```
t:satz;s:all;w:wort=haben
t:satz;s:all;w:id=16
t:satz;s:text;w:match=in den sauren Apfel beissen
t:satz;s:text;w:lemmata=sauer Apfel
```

**text:** Texte können anhand ihrer Id abgefragt werden. Als Ergebnis erhält man den ganzen Text.

Beispiel:

```
t:text;s:all;w:id=2
```

### Spezielle Kommandos im Servermodus

`cache.stat` gibt Cache-Statistik, `cache.keys` und `cache.items` geben den Inhalt aus. Bei `cache.items` wird die Ausgabe mit `_lr80` verkürzt. Achtung: Dies kann sehr lange dauern. Daher besser erst mit `status cache` über die Anzahl der Einträge informieren. `cache.clear` entfernt alle Einträge aus dem Cache.

`cache.autoclear mem <n>` stellt die Mindestschwelle zum automatischen Zurücksetzen des Cache auf `n` KB ein. Der Defaultwert ist in `config.py` definiert. `cache.autoclear time <m>` stellt ein, dass die Einhaltung der aktuellen Schwelle in etwa alle `m` Sekunden überprüft werden soll. Default: 600 s (= 10 min). Die aktuelle Schwelle wird bei jeder Prüfung halbiert, unterschreitet aber nie `n` KB. Wenn bei einer Überprüfung die Speicherbelegung des Servers über der aktuellen Schwelle liegt, wird `cache.clear` automatisch ausgelöst.

`info` zeigt Liste verbundenen oder kürzlich noch verbundenen Clienten. `info *` oder `info ID-Liste` gibt Details zum Clienten aus.

`status` gibt Antwort „OK“. `status server.start server.port dbuser dbname user usekorpora` gibt Details zum Server. `status *` gibt alle Details aus.

`shutdown` gibt Antwort „stopping service“ und beendet den Server. `quit` gibt Antwort „bye“ und schliesst die aktuelle Verbindung. Ein leeres Kommando (nur Whitespace) wird ignoriert.

## H.3.2 Beschreibung der Klassen

### querydb.py

In diesem Modul gibt es die Klasse `QueryBase`, die die Oberklasse für alle Anfragen an die Datenbank ist. Sie verarbeitet die Anfragen, und kann schon Fehler bei den Abfragen abfangen. Von ihr stammt die Klasse `QueryWort` ab, mit der man Anfragen an Woerter stellen kann. Die Klassen `QuerySatz` und `QuerySegment` stammen von `QueryPhrase` ab, da bei vielen Anfragen nur einige Spaltennamen in der Datenbank anders sind. Bei manchen Anfragen werden die häufigsten Wörter aus der Datenbank herausgefiltert. In der Klasse `Query` wird aufgrund der Anfrage ein Objekt der `QueryBase`-Klassen erzeugt, dass dann die Anfrage verarbeitet. War die Anfrage nicht an die Datenbank, sondern eine nach Status, Threads, dem Cache, etc., wird sie in dieser Klasse behandelt. Um die verschiedenen Ausgabeformate der Anfragen zu ermöglichen, gibt es die Klasse `Unpack` und deren Unterklassen. Die Klasse `QueryServer` stammt von `SocketServer` ab und ist der im Hintergrund laufende Server, der mittels Threads Anfragen an `Query`-Objekte abgibt.

### qclient.py

Analog zum Abfragen-Server ist dies ein Abfrage-Client, der Methoden für die verschiedenen Anfragen zur Verfügung stellt.

Man beachte, dass der Client durch Umstellung von `s:pyob` auf z.B. `s:text` unabhängig von den Modulen `DatabaseAPI` und `Region` gemacht werden kann.

### cache.py

Die Cache-Objekte verhalten sich prinzipiell wie normale Dictionaries mit der Ausnahme, dass sie von sich aus Key-Value-Paare löschen dürfen, um Platz für andere Daten zu schaffen. Zusätzlich bieten sie eine Zugriffskontrolle für Multithreading und führen eine Statistik über die Benutzung.

Wünschenswert wäre noch eine Funktion, dem Cache mitzuteilen, dass zu einem bestimmten Key in kürze der Value bereit steht. Wenn derzeit zwei Clienten kurz nacheinander die gleiche Anfrage stellen, führt der Server zweimal die gleichen Berechnungen durch: Der Cache kann erst gefüllt werden, wenn die Berechnung abgeschlossen wurde.

### config.py

In diesem Modul stehen viele Daten zur Konfiguration. Aufgrund des Usernamens bzw. der Umgebungsvariablen `KOKS_USER` und `KOKS_BIN_USER` werden Datenbankverbindungsdaten, Pfade und andere Werte für die Programme gesetzt. Soll ein neuer User hinzugefügt werden, muss in jedem Hash eine neuer Eintrag für diesen User gemacht werden.

### db.py

Das Modul `db.py` soll ein datenbankunabhängiger Wrapper für Datenbankaufrufe sein. Mit der Entscheidung für MySQL wurde aber die Datenbankunabhängigkeit nicht mehr getestet.

Im Modul gibt es Methoden, um Objekte der Klasse persistent zu machen, eine Methode, um den nächsthöheren Index für eine Tabelle zu finden, und Methoden, um Datenbankabfragen bzw. -änderungen vorzunehmen. Außerdem kann man eine Datenbankverbindung schließen, ein `commit` aussprechen, und sie zurücksetzen.

Auf Modulebene gibt es eine Methode, um einen Index hochzuzählen, und eine Methode, um eine Python-Long als String auszugeben, ohne den Buchstaben `L` am Ende (MySQL-Problem: die Indizes sind Longs, also bekommt man Python-Longs, als String repräsentiert haben diese immer die Form `1234L`, dies kann man aber nicht in die Datenbank eintragen).

### secret.py

`secret.py` berechnet den Einmalschlüssel für die Authentifikation beim Anfrageserver:

```
koks@gorina:~> telnet localhost 45628
Trying ::1...
telnet: connect to address ::1: Connection refused
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Was ist secret.getValue('Gqpnn90bCwR/IGdwJQ14BQ==')?
```

Aufruf von `secret.py`:

```
koks@gorina:~/align/DatabaseAPI> python secret.py 'Gqpnn90bCwR/IGdwJQ14BQ=='
HQ10VjAWPYzThopf7DIFsA==
```

Fortsetzung der telnet-Sitzung:

```
HQ10VjAWPYzThopf7DIFsA==
OK
status db
<2 items in list>
dbname = 'neul7b'
dbuser = 'koks'
<DONE>
quit
bye
<DONE>
Connection closed by foreign host.
koks@gorina:~>
```

**demo-qdb.py**

Diese Programm startet je nach übergebenem Parameter einen Query-Server oder einen Client, der auf einen laufenden Server zugreift. In `config.py` stehen verschiedene Ports, auf denen der Server laufen kann. Der Client versucht, wenn kein weiterer Parameter angegeben wird, der Reihe nach, sich auf diese zu connecten. Alternativ kann eine Portnummer angegeben werden. Will man z. B. einen Server auf dem zweiten angegebene Port starten, tippt man `demo-qdb.py -s 2`.

```
Usage: -client [ <port-key> ] | -server [ <port-key> ]
      key      port
      0        58597
      1        45628
      2        29295
      3        55776
      4        46581
      5        55149
```

**tagger.py**

Dies ist ein Python-Wrapper um den Tagger (s. Kapitel 6). Es liest den Pfad zum Tagger aus dem Skript `config.py`, und leitet dann den übergebenen Text an diesen weiter. Die Ausgabe des Tagger wird als Absatz-Objekt der Regionenschnittstelle zurückgeliefert.

**H.4 Korpora und die Datenbank****H.4.1 Eintragen der Korpora in die Datenbank**

Das Skript `korpus2db.py` trägt zwei alignte Korpora in die Datenbank ein. Es orientiert sich an den Markierungen `<segmentgrenze>` und `<SATZ>`, um die Segment- bzw. Satzgrenzen zu finden. Auch Markup innerhalb von `<KOKS></KOKS>` wird ignoriert.

Übergibt man dem Skript den Parameter `-e` und 2 Dateien, fragt es zunächst nach Angaben zu den Dateien und trägt sie dann als alignten Korpus in die Datenbank ein.

```
korpus2db.py -e <korpus1> <korpus2>
      -e: enter aligned korpora
```

**H.4.2 Löschen von Korpora**

Das Skript `delkorpus.py` kann zum Anzeigen und Löschen von Korpora benutzt werden. Übergibt man keine Parameter, listet es die eingetragenen Korpora und ihre ID aufgrund der Informationen in der Tabelle *Herkunft* auf. Übergibt man eine ID, so werden die Satz- und Segmentids des ersten und letzten Satzes herausgesucht und, wenn bei der Datenbankabfrage keine Probleme aufgetreten sind, diese gelöscht. Leere Korpora können deshalb nicht immer gelöscht werden - die minimale Plausibilitätsprüfung verhindert aber zumindest das Löschen der gesamten Datenbank.

# Literaturverzeichnis

- BAHNS, JENS (1996): *Kollokationen als lexikographisches Problem. Eine Analyse allgemeiner und spezieller Lernerwörterbücher des Englischen*. Tübingen: Niemeyer.
- BREIDT, ELISABETH (1995): "Extraction of V-N-Collocations from Text Corpora". In: *Proceedings of the Workshop on Very Large Corpora: Academic and Industrial Perspectives*. Ohio.
- BUTLER, C.S. (1985): *Systemic Linguistics. Theora and Applications*. London: Batsford Academic and Educational.
- COWIE, ANTHONY P. (1983): "General Introduction". In: *Oxford Dictionary of Current Idiomatic English*, herausgegeben von Cowie, Anthony P.; Mackin, Ronald und McCaig, I.R., Oxford: Oxford University Press, S. x–xvii.
- DUNNING, TED (1994): "Statistical Identification of Language". <http://citeseer.nj.nec.com/dunning94statistical.html>.
- FIRTH, JOHN R. (1957): "Modes of Meaning". In: *Papers in Linguistics 1934-1951*. London: Oxford University Press, S. 190–215.
- GALE, WILLIAM A. UND CHURCH, KENNETH W. (1993): "A program for aligning sentences in bilingual corpora". *Computational Linguistics* 19: S. 75–102.
- HAUSMANN, FRANZ JOSEF (1984): "Wortschatzlernen ist Kollokationslernen. Zum Lehren und Lernen französischer Wortverbindungen". *Praxis des neusprachlichen Unterrichts* 31: S. 395–406.
- KAY, MARTIN UND RÖSCHEISEN, MARTIN (1993): "Text-Translation Alignment". *Computational Linguistics* 19: S. 121–144.
- KJELLMER, GÖRAN (1987): "Aspects of English Collocations". In: *Corpus Linguistics and Beyond. Proceedings of the Seventh International Conference on English Language Research on Computerized Corpora*, herausgegeben von Meijs, Willem. Amsterdam: Rodopi, S. 133–140.
- KOBER, KATHARINA (2000): *Variation von Logotagmen*. Magisterarbeit, Universität Osnabrück.
- LEHR, ANDREA (1996): *Kollokationen und maschinenlesbare Korpora. Ein operationales Analysemodell zum Aufbau lexikalischer Netze*. Tübingen: Niemeyer.
- LUDEWIG, PETRA (2001): "LogoTax - un outil exploratoire pour l'étude de collocations en corpus". *tal (traitement automatique des langues)* 42:2. Special Issue on: Natural Language Processing and Corpus Linguistics / Traitement automatique des langues et linguistique de corpus.
- MANNING, CHRISTOPHER D. UND SCHÜTZE, HINRICH (1999): *Foundations of statistical natural language processing*. Cambridge, MA, London: MIT Press.
- RESNIK, PHILIP (1999): "Mining the Web for Bilingual Text". In: *Proceedings of the International Conference of the Association of Computational Linguistics*. Maryland.
- SCHMID, HELMUT (1994): "Probabilistic Part-of-Speech Tagging using Decision Trees". <http://www.ims.uni-stuttgart.de/ftp/pub/corpora/tree-tagger1.ps.gz>.
- SIBUN, PENELOPE UND REYNAR, JEFFREY C. (1996): "Language identification: Examining the issues". In: *Symposium on Document Analysis and Information Retrieval*. S. 125–135.
- SINCLAIR, JOHN (1978): "Collocation: a progress report". In: *Language Topics. Essays in Honour of Michael Halliday*, herausgegeben von Steel, R. und Threadgold, T. Amsterdam, Philadelphia: Benjamins, Band II, S. 319–331.
- VOSS, ANDREAS (2001): *Das große PC und Internet-Lexikon 2001 / 2002*. Düsseldorf: Data Becker.

# Index

- Überkreuzung, 48, 73
- Übersetzung, 5
  - kompositionelle, 12
- Üblichkeit, 11
- \_\_init\_\_.py, 347
- A\*-Algorithmus, 73
- Abkürzungen, *siehe* Satzendenerkennung
- absatz.py, 347
- absatzdb.py, 552
- abstand.py, 224, 372
- Abstandsmaß, 30
  - Übersicht, 65
  - Church und Gale, 55
  - in KoKS, 55
  - kombiniert, 63
  - Konzept, 53, 54
  - trigrammbasiert, 57
  - wörterbuchbasiert, 62
- Abstandsmatrix
  - Aufbereitung für Mavis, 71
  - beschleunigte Berechnung, 68
  - Darstellung, 72
  - in KoKS, 68
  - Kommandos, 70, 226
  - Konzept, 54
  - Sonderfälle, 68
- al.jar, 73, 228
- align.c, 230, 361
- align.sh, 70, 385
- align2al.py, 229, 386
- align\_1zu1.sh, 388
- align\_ai.sh, 71, 389
- align\_debug.sh, 390
- align\_mtr.sh, 71, 391
- align\_pur.sh, 392
- Aligner, 30
- Alignment, 48
  - Darstellung, 71, 72
  - Konzept, 54
  - Phrase, 6
  - Satzalignment, 6
  - Teilnehmersicht, 107
- Alignment-Pfad
  - Absatzbindung, 68
  - Darstellung, 72
  - Konzept, 54
- alignment.py, 228, 349, 392
- alignmentdb.py, 549
- Alignmentebene, 53
- alvis.py, 71, 228, 393
- Anfrageserver, 87, 257
- Authentifikation, 257, 260
- Funktionstest, 197
- Kommandos, 257, 259
- starten, 261
- Teilnehmersicht, 108
- Anwendung, 5
- Anwendungsszenario, *siehe* Demo-Applikation
- Architektur, 5
- atom.py, 349
- atomdb.py, 550
- Ausgangskollokationen, 30
- Authentifikation, 257
- Basis, 11
- Bedeutung, 7
  - kompositionelle, 8
  - wörtliche, 8
- Best Cell 1:1, 73
- Best-First-Suche, 73
- Bitext, *siehe* Korpus, paralleler
- Breidt, Elisabeth, 10, 12
- Browser, 91
- C&G Abstandsmaß, 57, 65
- cache.py, 259, 607
- CALL, 95
- Call
  - Szenario, 96
- CG Abstandsmaß, *siehe* C&G Abstandsmaß
- Chunk, 77
- Chunk-Parsing, 42, 77
- Church, Kenneth W., 55
- Church-und-Gale-Abstandsmaß, 55
- colligation, 9
- collocate, 9
- collocational span, 9
- config.pl, 215
- config.py, 196, 215, 260, 612
- Cowie, Anthony P., 11
- cqp, 78
- Dateiformat
  - Abstandinfo, 232
  - Abstandsmatrix, 231
  - Alignment, 232
  - Dokument, 44, 221, 231
  - Korpus-Verwaltung, 210
- Datenbank, 5, 85
  - Anlegen der Datenbank, 195
  - Anlegen der Tabellen, 256
  - Datenbank-Dump einspielen, 196
  - Eintragen der Korpora, 261

- Korpus löschen, 261
- Python-Schnittstelle, 256
- Wörterbuchzugriff, 256
- Datenbankdesign, 87
- db.py, 260, 617
- delkorpus.py, 261
- demo-a2.py, 394
- demo-abstand.py, 396
- Demo-Applikation, 6, 91
  - Installation, 196
  - Java-Applet, 91
  - markiertes Wort, 91
- demo-dok.py, 398
- demo-qdb.py, 261, 621
- DEP, 6
- Dictionary Entry Parsing, 6
- Document Type Definition
  - DTD, 25, 210
- Dokument
  - Dateiformat, 231
- dokument.py, 350
- dokumentdb.py, 553
- dokvis.py, 229, 398
- e-learning, 95
- eingabe.py, 71, 229, 399
- Einrichten der Software, 194
- EINS Abstandsmaß, 65
- Entropie, 23
- Environment Variablen, 196
- filterwb.py, 559
- Firth, John R., 9
- Fremdsprache, 4
  - Fremdsprachdidaktik, 10
  - Fremdsprachenlerner, 4, 5
- Frontend, *siehe* Demo-Applikation
- Funktionstest, 196
- Gale, William A., 55
- Gebrauchlichkeit, 7
- getalign.py, 228, 405
- getopt.c und getopt.h, 372
- Hausmann, Franz Josef, 11
- headingMapper.jar, 228
- Idiom, 8, 11
- IMS-Tagger, *siehe* Tagger
- Installation, 194
- Java-Applet, 91
- Java-Quellen übersetzen, 196
- Kjellmer, Göran, 10
- Kollokabilität, 9
- Kollokation, 7, 95
  - Kollokationsbegriff, 8
  - Kollokationsverständnis, *siehe* ebd.
  - significant collocation, 10
- Kollokations-Kandidaten, 30
- Kollokations-Prototypen, 30
- Kollokationsverständnis
  - statistisch orientiertes, 9
  - syntaktisch orientiertes, 11, 77
  - von KoKS, 12, 77
- Kollokator, 11
- Kombination
  - freie, 8
  - Wortkombination, 7
- Kombiniertes Abstandsmaß, 63
- Kompatibilität, 91
- Konfiguration, 196
  - Python zu Perl, 215
- Kontextualismus
  - Britischer, 9
- Kookkurrenz, 9
- Korpus, 5
  - Aufbereitung, 20
  - Bibel, 16
  - Browser, 25
  - DE-News, 16, 21
  - Eintragen in die Datenbank, 215, 261
  - EU, 17, 20
  - FR93, 78
  - KOKS, 24
  - Linux, 17
  - MCI, 17
  - NATO, 18
  - paralleler, 5, 16, 48
- korpus.py, 353
- korpus2db.py, 214, 261
- korpusdb.py, 553
- Korrespondenz, 53
- Kullback-Leibler Abstand, *siehe* Entropie
- L2-Lerner, 95
- Lemma, 11
- LemmatbuchVonDB (Klasse), 256
- Lemmatisierung, 42, 218
- Levenshtein-Algorithmus, 73
- Lexikographie, 10
- Lexikon, 10, *siehe* Wörterbuch
- matrix.py, 226, 406
- matrix.sh, 408
- Matrixdatei, *siehe* Dateiformat
- Mavis, 72, 228
- Mavis-Quellen übersetzen, 196
- mavis.jar, 228
- mavis.sh, 408
- Mehrwortlexem, 7
- mergemtr.py, 227, 408
- metatag-correction, 217
- Minimum Edit Distance, 73
- mogel.py, 541
- mtr2mavis.py, 71, 411
- Multiwortkonstruktion, 95
- MySQL, 85, 195
- n-Gramm, 23, 55, 60
- Nicht-Muttersprachler, 95
- nmatrix.py, 226, 412

- node, 9
- Normalisierung, 6, 37
  - Aufruf, 214
  - Ausgabeformat, 37
  - und Alignment, 37
- normalize.py, 214, 323
- NULL Abstandsmaß, 65
- OutOfMemoryError, 68
- Parsing, 42
  - Chunk-Parsing, 42, 77
- Phrase, 5
  - Alignment, 6
  - Erkennung, 6
  - Nominalphrase, 78
  - Phrasenhypothesen, 78
  - Präpositionalphrase, 78
  - Verbalphrase, 78
- phrasealign.py, 537
- Phrasenerkennung, 77
  - irrelevante Wörter, 77
  - Programm, 253
  - Vorgang, 80
- preproc.py, 214, 327
- Prototyp, 5
  - Demo-Applikation, 6
- punkt-tagger-english, 218
- punkt-tagger-german, 218
- py2pl.py, 215, 332
- Python
  - MySQLdb, 195
  - unterstützte Versionen, 194
- PYTHONPATH, 196
- qclient.py, 259, 603
- QueryBase (Klasse), 259
- querydb.py, 259, 568
- QueryServer (Klasse), 259
- region.py, 355
- regiondb.py, 548
- Relevant (Klasse), 257
- satz.py, 359
- satzdb.py, 551
- Satzendenerkennung, 218, 219
- Satzreihenfolge, 48
- secret.py, 260, 620
- segment.py, 360
- segmentdb.py, 552
- Selektionskriterium, 7
- separate-punctuation, 216
- Sinclair, John, 9
- Smoothing, 23
- Sonderzeichenrekonstruktion, 43, 217, 219
- Sprachidentifikation, 23
- Sprachlernen, 95
- Statistik
  - statistisch orientiertes Kollokationsverständnis, 9
  - statistische Verfahren, 5, 9, 10
  - statistisches Tagging, 42
- Tagfolgen, 77
- Tagger
  - Funktionstest, 196
  - Installation, 195
- tagger.py, 261, 622
- TAGGERHOME, 196
- Tagging, 6, 41
  - Part-of-Speech-Tagging, 42
  - Tree-Tagger, 42
- Tagset, 42
  - Penn-Treebank-, 42
  - STTS, 42
- TEST Abstandsmaß, 57, 65
- TG Abstandsmaß, 62, 65
- TG2 Abstandsmaß, 62, 65
- Tokenisierkorrektur, 217
- Tokenisierung, 216
- Translation Memory, 48
- tree-tagger, 218
- Trigramm, 57
- Trigrammbasiertes Abstandsmaß, 57
- Umgebungsvariablen, 196
- umlaut.pl, 217
- Unpack (Klasse), 259
- Visualisierung
  - Abstandsmatrix, 72
  - Alignment, 71, 72
- Wörterbuch, 30
  - Übersetzungsvariante, 32
  - Ausgabedateien, 31
  - Ausgangskollokationen, 30
  - Delimiter, 31
  - DEP, 30
  - Dictionary Entry Parser (DEP), 30
  - Fehler, 31
  - Format, 30
  - Kollokations-Kandidaten, 30
  - Kollokations-Prototypen, 30
  - Lexikoneintrag, 30
  - Mehrworteintrag, 30, 63
  - Quellen, 30
  - Schlüsselwort, 32
- Wörterbuchbasiertes Abstandsmaß, 62
- WB Abstandsmaß, 65
- wb.py, 554
- WB2 Abstandsmaß, 65
- WB2N Abstandsmaß, 65
- web\_phrases.py, 542
- WoerterbuchFilter (Klasse), 257
- WoerterbuchVonDB (Klasse), 256
- Wortart
  - bestimmung, 42
- Wortkorrespondenz, 53
- Wortstellung, 10
- Wortverbindung
  - Typologie von Wortverbindungen, 11
- zeitmessung.py, 417