KoMA

Korpus Matching Analysis

Michael Pach

Überblick

0. Hintergrund und Zielsetzung
1. Korpuslinguistik
2. Phraseologie/Idiomatik
3. Pattern Matching
4. Ähnlichkeiten in Texten
5. Reguläre Ausdrücke
6. KoMA

0 Hintergrund und Zielsetzung

Trend:

- Computereinsatz zur empirischen Forschung
- Sprachwissenschaftliche Forschung mit naturwissenschaftlichen Methoden
- Intelligente und angepasste Analyseverfahren

→ Leistungsfähige Rechner; Schnelle Massenspeicher; Geeignete Plattformen

Computerlinguistik (seit 1967):

Schnittstelle zwischen Informatik und Sprachwissenschaft

→ Automatisierte Untersuchung linguistischer Phänomene und Zusammenhänge
→ Aufstellung und Erforschung von Sprachtheorien
→ Informationsbeschaffung (Information-Retrieval) z.B. zur Textstellenfindung
→ Sprachverarbeitungssysteme zur Erleichterung der Mensch-Maschine-Interaktion

Zielsetzung von KoMA:

→ Syntaktische Analyse großer Texte (Korpora) auf Grundlage von Mustern (Token-Folgen)
→ Handhabung von Ähnlichkeiten und Toleranzen
→ Visualisierung, Auswertung und Ausgabe der Ergebnisse (Matchings)
→ Komfortable Plattform (realisiert in JAVA)

1 Korpuslinguistik

Zielsetzung:

- Beobachtung und Analyse des konkreten Sprachgebrauchs
- Verlässliche Aussagen bzgl. sprachlicher Phänomene

→ Repräsentative Sammlung von Texten (= **Korpus**) bildet die Forschungsgrundlage

Korpus:

- John McHardy Sinclair (1991): *„collection of naturally-occuring language text"*
- Verlangt gezielte Auswahl und Kategorisierung

→ Zeitungsartikel, Zitate, Korrespondenzen, ...
→ Interviews, Gespräche, Vorträge, ...

Intertextualität:

- Ulla Fix (2000): *„es ist unmöglich außerhalb der Welt der Texte und unabhängig von ihr zu kommunizieren"*
- Texte nehmen immer Bezug auf bereits produzierte oder konsumierte Texte
- Unterschiedlichste Ausprägungen: Adaption, Anspielung, Quelle, Zitat, ...

→ Gérard Genette (1982): **Transtextualität** (*„Texte zweiter Stufe")*

1.1 Computer-/Korpora

Korpora:

- **Oxford English Dictionary** (London, seit 1857)

  Von Hand ausgesuchte Zitatesammlung (heute über 4 Millionen Zitate)
- **NEGRA-Korpus** der Universität Saarbrücken

  Sammlung deutscher Zeitungstexte der Frankfurter Rundschau (20.000 Sätze)

Computerkorpora:

- **Brown Corpus** (Brown University, 1964-67)

  500 amerikan. Texte des Jahres 1961 aus 15 Kategorien
- **LOB-Korpus (**Universit. Lancaster, Oslo, Bergen, 1970-78)

  Orientierung am Brown Corpus, jedoch mit britischen Texten

➔ Vergleichsmöglichkeit englischer und amerikanischer Texte

Kategorie	Genre
A	Press: reportage
B	Press: editorial
C	Press: reviews
D	Religion
E	Skills, trades and hobbies
F	Popular lore
G	Belles lettres, biography, essays
H	Miscellaneous
J	Learned and scientific writings
K	General Fiction
L	Mystery and detective fiction
M	Science fiction
N	Adventure and western fiction
P	Romance and love story
R	Humour

- **IDS-Mannheim** (70 Millionen Wortformen) mit Recherchesystem

  COSMAS; **AGD** mit ca. 900 Videos und 16300 Tonaufnahmen
- **BNC/ANC** (seit 1991) - 100 Millionen Einträge untersch. Bereiche
- **BOE** (seit 1980) - Cobuild-Projekt (Leitung: John Sinclair)

  300 Millionen Wortformen gesprochener und geschriebener Sprache

1.2 Verarbeitungsmethoden

Auswahl:

- Repräsentative und ausgewogene Auswahl für allgemein gültige Aussagen
- Abdeckung eines breiten Spektrums sprachlicher Kommunikation

→ Verlangt Kategorisierung und Klassifizierung

Aufbau:

- Zerlegung in kleinste definierte Informationseinheiten (**Tokens**): Wörter, Halbsätze, Sätze, ...

  Bsp: *„|Hunde| |die bellen| |beißen nicht|"*

- **Annotierung:** Zuweisung von Zusatzinformationen. Meist über **Tags** (<Merkmal>)

  Bsp: *Tag-Set: <Position, Wortstamm, Wortart> lief <4, laufen, Verb>*

Hilfsmittel:

- **Parsing:** Syntaktische Analyse von Abhängigkeiten. Fast annotierte Tokens zu größeren syntaktischen Einheiten zusammen
- **Indizierung:** Z.B. über Frequenzlisten, die das Tokenvorkommen in bestimmten Textabschnitten protokollieren (Position, Häufigkeit)
- **Konkordanzen:** Auflistung und Sortierung wichtiger Wörter (Schlüsselwörter)

1.2 Verarbeitungsmethoden

Datenbasis:

- Unstrukturierte Rohtext-Dateien
- Strukturierte Texte (SGML, XML)
- Datenbanksysteme (SQL, DB2, ...)

Beispiel:

Sonderforschungsbereich 441 (Eberhard-Karls-Universität Tübingen, seit 1999)

- Methoden- und phänomenorientierte voneinander abhängige Projekte
- Datenbasis: TUSNELDA-Sammlung (Datenbank-basierte annotierte Korpora)
- Manuelle oder halbautomatische Datensatzerstellung (z.B. über Fragebögen)
- Themen: Beste syntakt. Strukturen, Semantik/Intuition, starke/schwache Variablen, ...

KoMA's Unterschiede:

- Korpus: Einzeltexte oder Textsammlung für individ. oder prinzipielle Untersuchungen
- Datenbasis: Text-Dateien zur flexiblen Handhabung von Rohtexten
- Bausteine: Tokens (temporär) zur strukturellen automatischen Vorkommensanalyse
- Aufbereitung: Syntaktische Filterung; keine Annotierung
- Verarbeitung: Token-Matching und strukturierte Speicherung der Ergebnisse (XML)

2 Phraseologie/Idiomatik

Definitionen:

Ulf Abraham (1982): *„Formelhaftes Reden/Schreiben zur verkürzten Darstellung von Sachverhalte" [...] „Nutzung des verfügbaren gesellsch. Wissens und kollektiver Erfahrungen"*

- **Phraseme/Phraseologismen** (griech. „Phrase"): Feste, konventionelle, meist bildliche Wortverbindungen, als Bestandteile des täglichen (traditionellen) Sprachgebrauchs
- **Idiome** (griech. „Eigentümlichkeit"): Tiefere Bedeutung/Aussage des Phrasems

  Bsp: *„die Katze im Sack kaufen"* *[etwas erwerben ohne zu prüfen]*

Kriterien:

- Mindestens ein **Autosemantikum** und ein **Synsemantikum**
- **Reguläre** (wörtlich Sinn ergeben) oder **irreguläre** Wortgruppe („Bahnhof verstehen")
- Unterscheidung von un-/durchsichtigen und voll-/teil-idiomatischen Phrasemen

1) **Reproduzierbarkeit** und **Lexikalisierung** – **F**estes, unveränderliches Sprachgut
2) **Idiomatizität** - Idiomatische Umdeutung des Phrasems mit z.T. historischem und kulturellem Hintergrund („seinen Friedrich Wilhelm darunter setzen")
3) **Nicht-Satzwertigkeit** - Fungierung als Satzglieder („alt aussehen") oder als Satzanhängsel („im Prinzip") - Ausnahmen: Sprichwörter, geflügelte Worte und Ähnliches („carpe diem")

2.1 Funktionen und Klassifizierung

Funktionen:

- **Begrüßung, Einleitungen** („Herzlich Willkommen“)
- **Aufruf, Parolen, Slogans** („dalli dalli“, „Brot für die Welt“)
- **Begriffe, Phrasenkomposita** („status quo“, „Eis am Stiel“)
- **Vergleiche, Bewertung** („stumm wie ein Fisch“, „super“)
- **Verhalten, Tätigkeit** („Leine ziehen“)
- **Geschehnisse** („den Löffel abgeben“)
- **Weisheiten, Aussagen** („Unrecht Gut gedeihet nicht“)

Kategorien: (nach Burger, Buhofer, Sialm / Coulmas)

- **Phraseologische Ganzheiten** („jmd. über die Schulter schauen“)
- **Phraseologische Verbindungen** („kalter Krieg“)
- **Phraseologische Vergleiche** („arm wie eine Kirchenmaus“)
- **Phraseologische Termini** („das rote Kreuz“)
- **Modellbildungen** („Schritt für Schritt“)
- **Funktionsverbgefüge** („Zeugnis ablegen“)
- **Zwillings-/Drillingsformen** („heimlich, still und leise“)
- **Sprichwörter/Antisprichwörter** („der Apfel fault nicht weit vom Stamm“)
- **Gemeinplätze** („was sein muss, muss sein“)
- **Routineformeln** („Entschuldigung“)

3 Pattern Matching

Motivation:

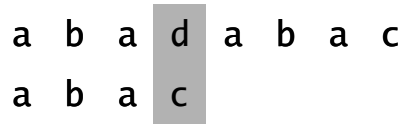
- Finde in einer Zeichenreihe (Länge n) über einem Alphabet ∑ ein oder alle Vorkommen eines gegebenes Musters (Pattern) mit Länge m
- Ziel: Optimierung der Suche durch geeignete Algorithmen

→ Vollständiges Auffinden des Musters (hartes Matching)

Strategien:

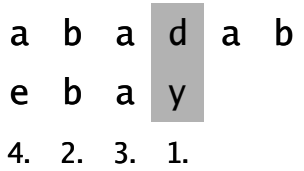
Symbolvorkommen

Ermittle in Vorlaufphase (Preprocessing) die längst mögliche Schiebedistanz - Lege Tabelle an, die für jedes Symbol aus ∑ die Position des letzten Vorkommens im Muster aufzeigt (O(m))



Wahrscheinlichkeiten

Vergleiche Symbole in aufsteigender Wahrscheinlichkeit ihres Auftretens
Sortierte Tabelle, die jedem Mustersymbol dessen Auftrittswahrsch. zuweist (O(m log(m)))



Signaturen

Fasse Muster/Zeichenfolge zur einer eindeutigen Signatur (z.B. Hash) zusammen, die anstelle des Musters verglichen wird. (O(1)) - Inkrementelle Berechenbarkeit (O(m) bzw. O(n))

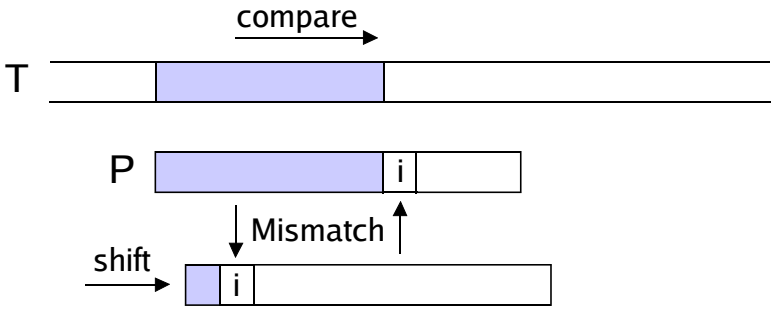
3.1 Brute Force

Methode:

- Prüfe das Muster symbolweise von links nach rechts, bis zum ersten Mismatching
- Shift um eine Position

→ Worst Case von O(nm)

→ Im Mittel O(2n) Vergleiche; O(n) bei großem ∑



Variante 1:

- Start des Mustervergleichs beim zweiten Symbol (2, ... , m-1, m, 1).
- Bei Matching, Shift um zwei Positionen, sonst um eine Position
- Setzt Muster mit mindestens zwei verschiedenen Zeichen voraus

→ Mittlere Laufzeit: O(n)

Variante 2:

- Vergleiche das Muster in der Reihenfolge der wahrscheinlichen Zeichenvorkommen
- Preprocessing: Sortierte Tabelle mit Wahrscheinlichkeiten O(m log(m))

→ Mittlere Laufzeit: O(n)

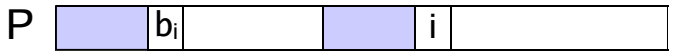
3.2 Knuth-Morris-Pratt

Preprocessing:

- Finde breitesten Rand von P
- Finde für jedes Muster-Präfix (0, ... ,i-1) dessen breitesten Rand b[i]
- b[i] erhöht sich inkrementell bei Fortsetzbarkeit

→ Shift: |Präfix| - |Rand|

→ Laufzeit: O(m)

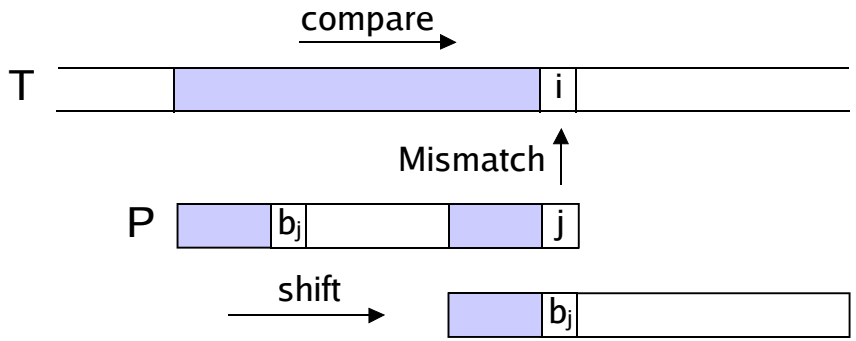



```
int i = 0, j = -1;
b[i] = j;
while (i < m){
    while (j >= 0 && p[i] != p[j])
        j = b[j];
    i++;
    j++;
    b[i] = j;
}
```

Algorithmus:

- Ähnlich wie das Preprocessing – Hier: Finde Rand P in Text-Präfix mit Länge m
- Vergleiche Zeichen der Reihe nach von links nach rechts
- Tritt an Position t[i] bzw. p[j] ein Mismatch auf, Shift um (j-1)-b[j]

→ Laufzeit: O(n)



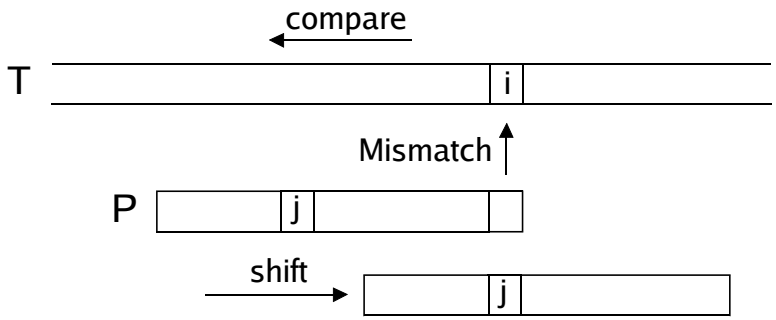

```
int i = 0, j = 0;
while (i < n){
    while (j >= 0 && t[i] != p[j])
        j = b[j];
    i++; j++;
    if (j == m){
        hit(i-j);
        j = b[j];
    }
}
```

3.3 Boyer-Moore

Bad-Character-Heuristik:

- **Preprocessing**
  Ermittle für jedes Musterzeichen dessen letztes Vorkommen.
- → Nutze indiziertes Array mit Länge $|\Sigma|$
- → Laufzeit O(m)

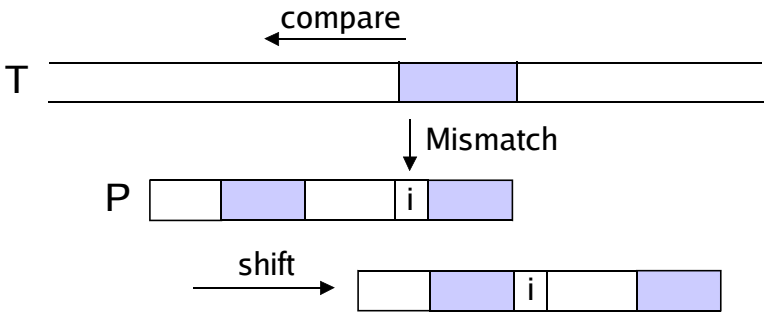
- **Algorithmus**
  Vergleiche zeichenweise von rechts nach links bis zum Mismatch an t[i].
- → Shift um (i – j) nach rechts
- → Laufzeit: O(n/m)
- → Worst Case: O(nm)



Good-Suffix-Heuristik:

- **Preprocessing**
  Suffix von P bis zum Mismatch kann in P erneut vorkommen - Finde für jedes Suffix dessen breitesten Rand (ähnlich KMP)
- → Laufzeit O(m)

- **Algorithmus**
  Vergleiche wie bei Bad-Char.-Heuristik
- → Shift bis Deckungsgleichheit
- → Pointer um (m+1) - |Shift| nach rechts
- → Laufzeit: O(n)
- → Verhindert worst case O(nm) bei BC-Heur.



→ **Shift um Maximum(Heuristiken) - Im Mittel O(n/m) Vergleichsoperationen**

3.4 Andere Verfahren

Quick Search / Hoorspool / Sunday:

- Nutzen Boyer-Moore-Algorithmus nur mit Bad-Character-Heuristik
- → Vorlaufphase: O(m)
- → Suchphase: O(n) bis O(n/m)

Colussi:

- Durchläuft das Muster in zwei Phasen und verwendet KMP-Preprocessing
- Phase 1: Durchlaufe das Muster von links nach rechts und vergleiche diejenigen Zeichen mit b[j] ≥ 1 (**noholes**). Shift gemäß KMP bei Mismatch
- Phase 2: Vergleiche die übrigen Stellen (**holes**) von rechts nach links - Bei Mismatch, Shift um das längste Suffix, das mit Muster-Präfix übereinstimmt
- → Vorlaufphase: O(m)
- → Suchphase: O(n)

Skip Search:

- Lege für jedes Alphabetsymbol ein Bucket (Stack) an, das die Positionen der Mustersymbole auflistet. (|∑| viele)
- Suchphase prüft Bucketeinträge anstelle des Musters
- → Vorlaufphase: O(m) und O(|∑|) Platz
- → Suchphase: O(n) (worst case: O(nm))

4 Ähnlichkeiten in Texten

Motivation:

- Finde das best mögliche Matching eines Musters in einer Zeichenreiche (über Alphabet ∑) **Syntaktische Ähnlichkeit**
- Ziel: Minimierung der Unterschiede

S	c	h	i	l	l	e	r
|	⊥	⊥	_	|	⊥	|	|
S	p	a		l	i	e	r

→ Teilweises Auffinden des Musters (weiches Matching)

→ Ansätze: **Editierdistanz**, **k-Differences**, **Sequence Allignment**

1) Editier/Levenshtein-Distanz:

- Überführe über elementare Editieroperationen (Löschen, Einfügen) eine Zeichenreihe (A) in eine andere (B)
- Finde den kürzesten Weg (trace) zur Transformation und berechne Gesamtkosten D(A, B) (Einheitskosten-Modell)
- Je Zeichen maximal eine Veränderung: O(max(|A|, |B|))
- Jede Zelle der Distanzmatrix minimiert die lokalen Kosten

		S	**c**	**h**	**i**	**l**	**l**	**e**	**r**
	0	1	2	3	4	5	6	7	8
S	1	0	1	2	3	4	5	6	7
p	2	1	1	2	3	4	5	6	7
a	3	2	2	2	3	4	5	6	7
l	4	3	3	3	3	3	4	5	6
i	5	4	4	4	3	4	4	5	6
e	6	5	5	5	4	4	5	4	5
r	7	6	6	6	5	5	5	5	4

- Bottom-Up-Prinzip: **D(a,c) ≤ D(a,b) + D(b,c)**
- D[|A|,|B|]: **Levenshtein-Distanz**

→ Laufzeit: O(|A||B|)

→ Platz: O(|A||B|); O(min(|A|, |B|) matrixoptimiert

D[i,0] = i, i = 1..|A|
D[0,j] = j, j = 1..|B|

D[i,j] = min(D[i-1,j-1] + c(A[i],B[j]),
D[i-1, j] + 1,
D[i, j-1] + 1),
i = 1..|A|, j = 1..|B|

4.1 Syntaktische Ähnlichkeiten

2) k-Differences:

- Gesucht sind alle Matchings von B in A mit bis zu k Unterschieden
- Ähnlich Levenshtein: Jeder Matrixeintrag enthält die Editierdistanz der zu [b_1 ... b_i] ähnlichsten Subsequenz [a_j' ... a_j] Andere Initialisierung
- Monoton steigende Distanzwerte: Subtraces

→ Laufzeit: O(|A||B|) Platz: O(|A||B|); O(|A|) (nur Position)

		A	C	E	A	B	P	C	Q	D	E
	0	0	0	0	0	0	0	0	0	0	0
A	1	0	1	1	0	1	1	1	1	1	1
B	2	1	1	2	1	0	1	2	2	2	2
C	3	2	1	2	2	1	1	1	2	3	3
D	4	3	2	2	3	2	2	2	2	2	3
E	5	4	3	2	3	3	3	3	3	3	2

E[i,j] = min(B[i], A[j'..j])
E[i,0] = i, i = 1..|B|
E[0,j] = 0
E[i,j] = analog Levenshtein

3) Sequence Allignment:

- Zwei Sequenzen A und B sollen mit bestmöglicher Übereinstimmung aneinander gelegt werden:

A G C _ G T T _ A G G
A G C A C T T A A G G

- Maximiere Wertrechnung S(A, B) (Score)
- Werte: Match +1, Mismatch -1, Leerzeichen -2
- Initialisierung der mit max. Präfixkosten
- Trace ergibt optimales Alignment

→ Laufzeit: O(|A||B|); Platz: O(min(|A|, |B|) (für Gesamtkosten)

		C	A	A	G
	0	-2	-4	-6	-8
C	-2	+1	-1	-3	-5
A	-4	-1	+2	0	-2
A	-6	-3	0	+3	+1
A	-8	-5	-2	+1	+2
G	-10	-7	-4	-1	+2

S[i,0] = -i*cost, i = 1..|A|
S[0,j] = -j*cost, j = 1..|B|

S[i,j] = max(S[i-1,j-1] + s[i,j],
S[i-1, j] - cost,
S[i, j-1] - cost),
i = 1..|A|, j = 1..|B|

4.2 Phonetische Ähnlichkeiten

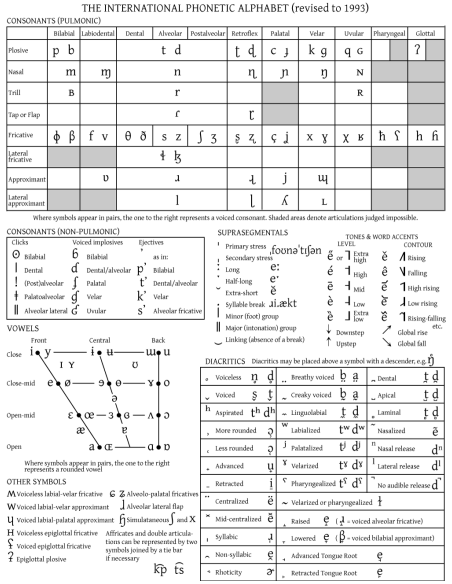
Motivation:

- Lautfolgen zur Ähnlichkeitsbehandlung (*Bsp: sea* / *see*)
- Orientierung am menschlichen (subjektiven) Sprachverständnis

→ Erfordert spezifische Lautsymbolik zur Handhabung

→ Transkriptionsmethoden (Zeichen → Lautsymbol)

→ Vergleichsmethoden für Laute (Phone)



Lautsymbolik:

- **IPA** (IPA, 1888): „Computer“ → [kʰ m’pju:tʰ (r)]
- IPA in Unicode: „Spiel“ → U+032F U+0283 p i U+02D0 l
- **X-SAMPA** (1995): „plötzlich“ → [’pl9tslIC] (IPA in 7Bit-ASCII)

Vergleichsmethoden:

1) **Identitätsabgleich**: Pattern-Matching/Distanzmessung (Levenshtein) über Lautsymbole
2) **Covington** (1996): Punktesystem zur bewertung von Konsonanten und Vokalen (Lautlänge, Sprünge) - Je nach Abweichung werden Strafpunkte vergeben
3) **Hartman** (1981): Umfangreiche binäre Charakterisierung von Lautmerkmalen - Unterscheidet auch Tonhöhen und Ausdrucksformen (gehaucht, hart)
4) **Kondrak** (2003): Distanz zweier Phone ergibt sich aus Summe gewichteter Lautmerkmale

5 Reguläre Ausdrücke

Motivation:

- Flexible Suche nach Mustern in Zeichenketten
- Behandlung syntaktischer Ähnlichkeiten (Pseudoähnlichkeit) mit Hilfe kompakter Ausdrücke

  Bsp: *„Haus[a-z]*“* *findet Haus, Hausfrau, Haushalt, Hausboot, ...*

Formale Sprachen und Grammatiken:

- Formale Sprachen werden über ein Regelwerk (Grammatik) beschrieben
- Natürliche Sprachen $\subset$ Formale Sprachen (math. Objekte)
- **Grammatik:** 4-Tupel mit (V: Variablen, Σ: Terminale, P: Prod.regeln, S: Startsymbol aus V)

  Bsp: $S \rightarrow aB; B \rightarrow b \mid bC; C \rightarrow c \mid B$ *(Sprache beinhaltet; ab, abc, abb, ...)*

Chomski-Hierarchie (1956):

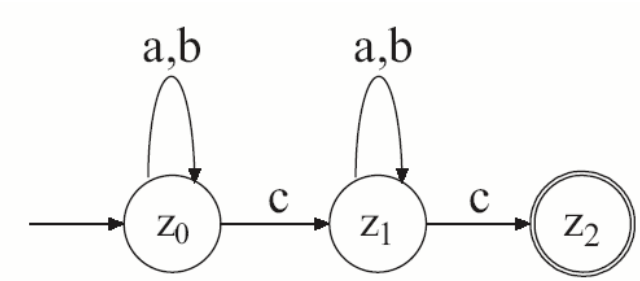
Sprach/Grammatik-Typen mit zunehmender Einschränkung des Regelwerks:

- Typ-0 (allgem. Phrasenstrukturgrammatik): Keine Einschränkungen; rekursiv aufzählbar
- Typ-1 (kontextsensitive Grammatik): |Konklusion| ≥ |Prämisse|
- Typ-2 (kontextfreie G.): Typ-1; Prämisse nur Variablen; Konklussion mit Variablen und Termini
- **Typ-3 (Reguläre Grammatik):** Typ-2; Rechtslinearität ($V \rightarrow t$ oder $V \rightarrow tV$)

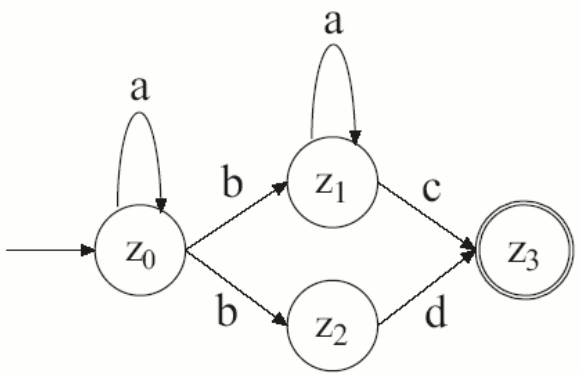
5.1 Endliche Automaten

- **Sprachakzeptor:** Entscheiden, ob ein gegebenes Wort zu einer Sprache passt
- **(Transducer):** Definiert zu den Folgezuständen noch Ausgabe (Einsatz bei Schaltwerken)
- **DEA:** 5-Tupel mit (Z: Zustände, Σ: Alphabet, δ: Z x $\Sigma \rightarrow$ Z, z0: Startzustand, E: Endzustände)
- **NEA:** DEA mit S: Menge von Startzuständen und δ: Z x $\Sigma \rightarrow$ P(Z)

*Bsp: (a|b)*c(a|b)*c als DEA*



*Bsp: (a)*b(((a)*c)|d) als NEA*



- NEA $\rightarrow$ DEA: Jede mögliche Teilmenge von Zuständen als Einzelzustand (+ Minimierung)
- Reguläre Ausdrücke (Formelwerk) beschreiben reguläre Sprachen (definiert über reguläre Grammatiken), die von NEA/DEA akzeptiert werden

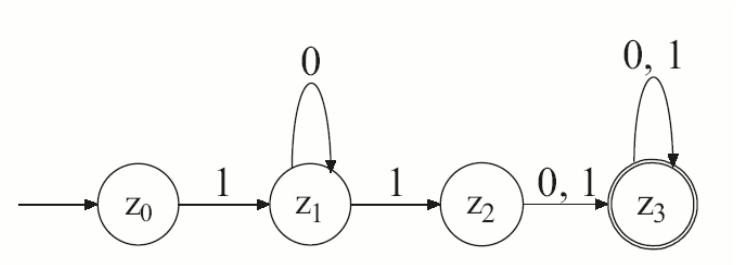
→ **Implementierung von EAs zur Zeichensuche über reguläre Ausdrücke**

5.2 Syntax

Handling regulärer Ausdrücke:

- Parsing des regulären Ausdrucks
- Generierung des EA:
  Platzbedarf: DEA O(2^m); NEA O(m)
- Akzeptanzkontrolle der Zeichenfolge:
  NEA: Über Backtracking O(nm)
  DEA: O(n)

*Bsp: 1(0)*1(0|1)(0|1)* als DEA*

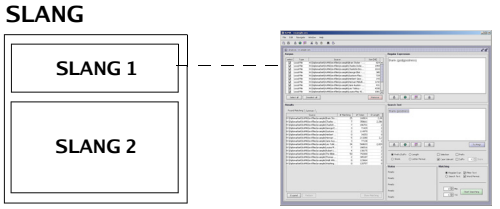


Operationen (java.regex):

• Verkettung	Ruderboot	Ruderboot
• Alternativen	ist | war	ist, war
• Beliebige Einzelsymbole	.i..en	sieben, biegen, ...
• Kleen'scher Abschluss	r(i)*esig	rsig, riesig, riiesig, ...
• 1-/N-maliges Vorkommen	(Z)+eitung	Zeitung, ZZeitung, ...
• 0-/1-maliges Vorkommen	L(a|u)?mpe	Lmpe, Lampe, Lumpe
• Zeichenklassen	[0-2]*	2, 021, 1, 021022, ...
• Vorkommen/Ausschluss	[^ad]{3}	bbc, cbc, ebc, fbc, ...
• Metazeichen	\s, \S (= ^\s)	Leerzeichen, Platzhalter
	\d, \D	Zahlen [0-9], ^Zahlen
	\w, \W	[0-9a-zA-Z], [^0-9a-zA-Z]

6 KoMA

Rahmen:



- **KoMA:** Eigenständiges Tool / Modul von SLANG1
- **SLANG** (SLANG1 / SLANG2 - Wilhelm-Schickard-Institut):
  Modulares Projekt zur syntaktisch, semantisch und pragmatischen Sprachuntersuchung
- SLANG1: Textaufbereitung mit Zerlegung in Informationseinheiten (Tokens) - Datenbasis: SQL-Datenbank
- SLANG2: Bedeutungsanalysen (Akteure, Dialoge, Thematik, ...) auf Basis von Tokens und Äußerungseinheiten (Illocution Units)

Suchprinzipien:

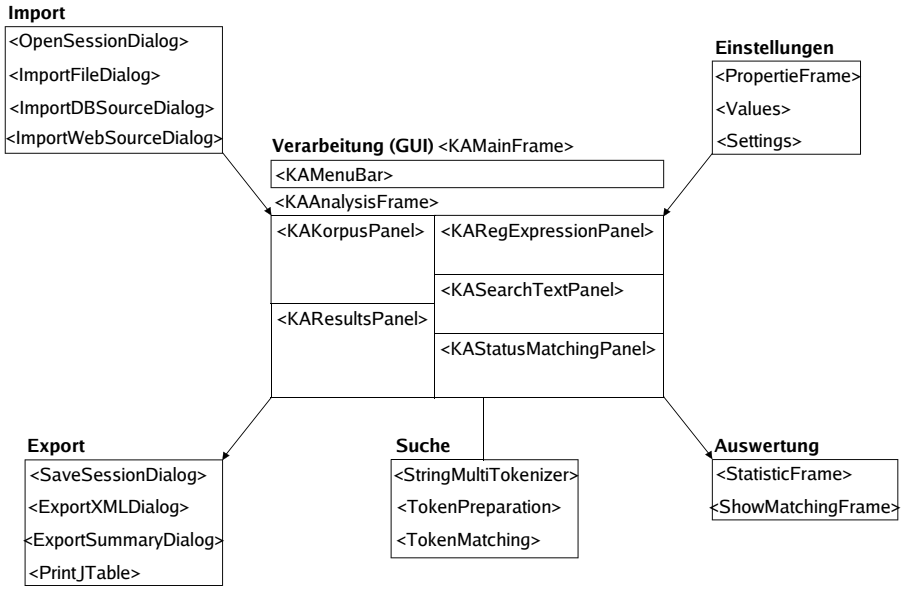
- Suche nach Tokenketten, deren Länge über einem Schwellenwert liegen
- Statt Pattern-Matching → Token-Matching ($\sum$ = Menge von Tokens)

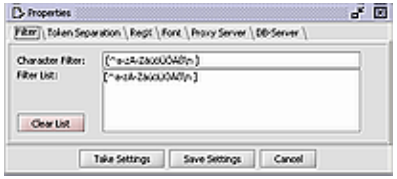
- **Token-Matching/Ähnlichkeit** (über reguläre Ausdrücke)
  Gesucht: *„Die Stille wurde unterbrochen."* Gefunden: *„die Stillen wurden"*
- **Token-Permutation** (Teilmengen)
  Gesucht: *„Sag niemals nie"* Gefunden: *„sag nie"*
- **Token-Toleranz**
  Gesucht: *„Ich gehe jetzt nach Hause"* Gefunden: *„ich gehe nach Hause"*
  Gefunden: *„ich gehe jetzt sofort nach Hause"*

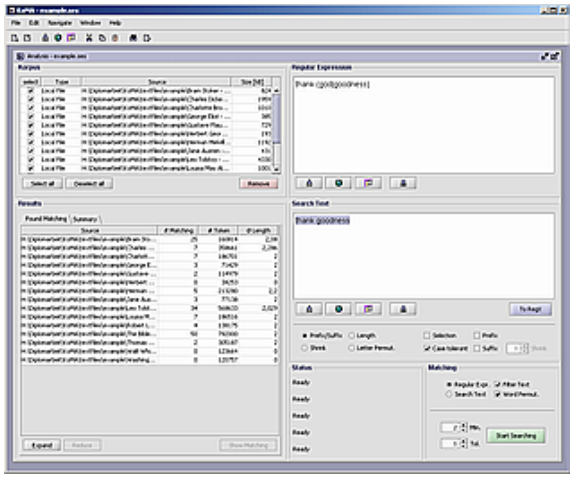
6.1 Aufbau

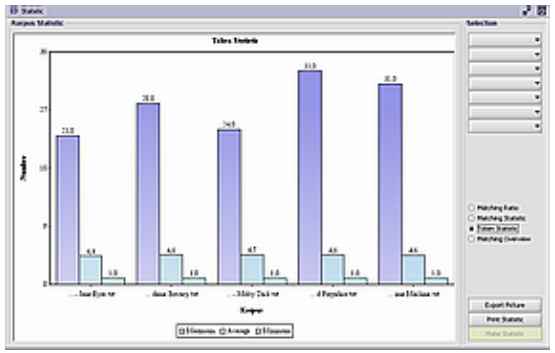
- Sprache: Java (SDK 1.4.2)
- Entwicklungsumgebung: Eclipse
- GUI: AWT/Swing

- Modularer Aufbau: „Form folgt Funktion”
- Module (Internal Frames/Dialoge/Panels) z.T. Threads zur parallelen Ausführbarkeit









6.2 Funktionalität: Daten-I/O

Korpus: (<KAKorpusPanel>)

- Quellen: Lokale Files, Web-Quellen, SLANG1-SQL-Datenbank
- Formate: ISO/IEC-8859-1 (8-Bit), Unicode (UTF-8, UTF-16, UTF-32)

Suchtexte: (<KASearchTextPanel>, <KARegularExpressionPanel>)

- Import: Korpus-Quellen/Formate
- Export: Inhalt der Textpanels (Tokens regulärer Ausdrücke) in Unicode (UTF-16)

Sessions: (<Import/Export-SessionDialog>)

- Speichert gegenwärtigen Stand einer Suche im GZIP-Format
- Beihaltet: Einstellungen, Suchtexte, Ergebnisse, Korpora (nur deren Links)

Settings / IDs: (<Settings>, <Values>)

- Import/Export der Grundeinstellungen (Filter, Sprache, Schrift, Proxy, ...) in [settings.txt]
- Sprachdatei (*.ids) zur KoMA-Anpassung (Englisch, Deutsch)

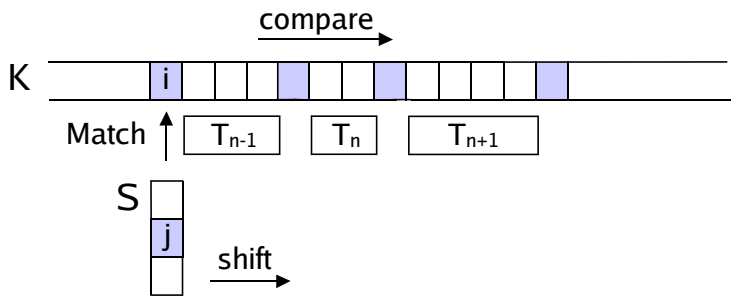
Results: (<ExportSummaryDialog>, <ExportXMLDialog)

- Export (textuell) des Suchprotokolls (Dauer, Einstellungen, Token-/Matchingauswertung, ...)
- Export (strukturiert) sämtlicher Matchings (Position, Länge, ...) und Statistiken im XML-Format

6.2 Funktionalität: Tokenizer/Filter

Token-Separation:

- Zerlegt Korpus/Suchtext in Vergleichsbausteine
- Korpus: Token → natürlicher Text
- Suchtext: Token → regulärer Ausdruck



- Unterstützt werden bis zu 3 Separatoren
- Sequentielle Suche (indexOf(*Separator*)) mit O(n/min(m)) Suchoperationen

„Haus und Hof" → *„|Haus| |und| |Hof|"* (Separator: *„ "*)
→ *„|Haus | | Hof|"* (Separator: *„und"*)

Stringfilterung:

- Vor dem Token-Matching werden aus Korpora und Suchtext unerwünschte Zeichen (z.B. Satzzeichen „“,.) oder Zeichengruppen (z.B. Versangaben [Gen:1:17]) herausgefiltert
- Ersetzung durch Füllzeichen (Backspace: **\b**), die beim Tokenvergleich ignoriert werden
- Filterdefinition über regulären Ausdruck (Standard: [^a-zA-ZäüöÜÖÄß\n])

Filter: [^a-zA-ZäüöÜÖÄß\n]
„[langsam], (langsam)"* → *„langsam langsam"*

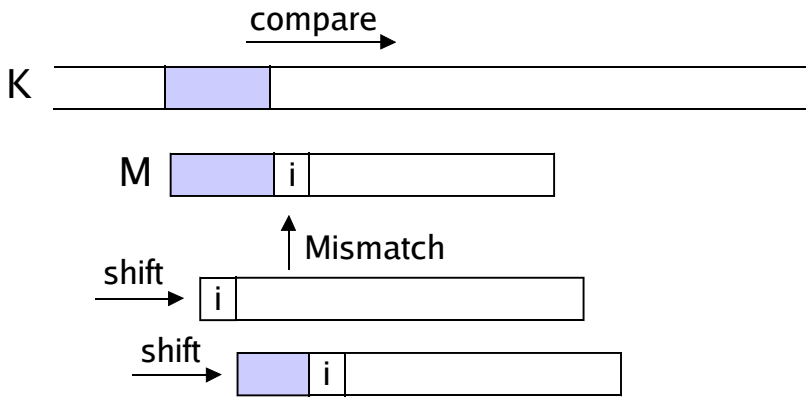
- KoMA prüft korrekte Syntax regulärer Ausdrücke - Bei Fehler: Einsatz von Platzhalter: [\S]*

*„Ende gut, alles]*gut"* → *„Ende gut, alles [\S]*"*

6.2 Funktionalität: Token Matching

Algorithmus:

- **Normales Matching:** Brute Force mit Abbruch bei Mismatch → Laufzeit: O(n)
- **Token-Permutation:** Brute Force mit Abbruch bei Matching → Laufzeit: O(nm)




```
int i = 0, j = 0;
while (i <= n - m){
    while (j <= m && k[i] == p[j])
        i++; j++;

        if (j >= b)
            hit(i – j);

        i = i - j + 1;
        j = 0;
}
```

Gründe:

- Token-Matching über reguläre Ausdrücke macht Preprocessing unmöglich
- → **Knuth-Morris-Pratt ungeeignet**

  Pattern: *„mit M... und M[\S]*"*

  Korpus: *„mit Mann und Maus"* → *„M..."* ≠ *„M[\S]*"* matchen *„Mann"*, *„Maus"*

- Suche nach Präfixen setzt Suchrichtung von links voraus und |∑| sehr groß (bzw. unbekannt)
- → **Boyer-Moore ungeeignet**

- Da |∑| sehr groß: Laufzeit linear → Abbruch meist beim 1.Token (∅ Matching: 3 Token)

6.2 Funktionalität: Ähnlichkeiten

<KARegularExpressionPanel>

Manuelle Eingabe von Tokens regulärer Ausdrücken beliebig kombinierter Operationen:

„(E|e)in Unglück kommt (selten|immer) allein" (Alternativen)
„[Ee]s ist nicht alles [A-Z]old was glänzt" (Zeichenklassen)
„Leben wie G.tt in .....reich" (Beliebige Einzelsymbole)

<KASearchTextPanel>

Klartexteingabe mit automat. Umformung in reguläre Ausdrücke aus vier Ähnlichkeitsklassen:

- **Präfix/Suffix:** *„[\S]*schritt"* → *„Fortschritt"*
  „Haus[\S]"* → *„Haushalt"*
- **Token-Länge:** *„.{4}"* → *„Boot", „Haus"*
- **Token-Präfix:** *„Men[\S]*" (3)* → *„Mensch"*
- **Zeichen-Teilmenge:** *„[Zimmer]*"* → *„Zier" , „immer" , „mir"*

Tokenvariationen:

- **Token-Permutation/Teilmenge**
  Muster: *„Sich regen bringt Segen"* Korpus: *„Segen bringt Regen"*
- **Token-Toleranz**
  Muster: *„gefällt mir gar nicht"* Korpus: *„gefällt mir ganz und gar nicht"* (+2)
  Korpus: *„die Sache gefällt mir _ nicht"* (-1)

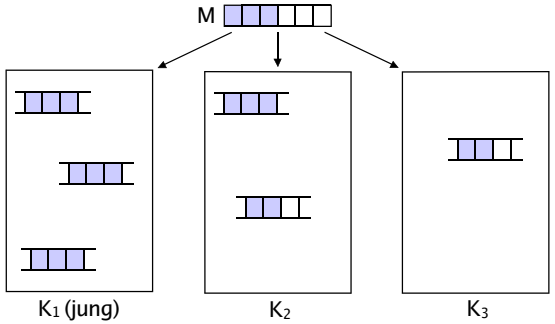
6.3 Aussagen

Herkunft / Urheberschaft / Genre

- Phraseme: Lassen durch ihre sprachliche Beständigkeit thematische Schlüsse zu
  „es war einmal" (Märchen, Fabeln) *„nennt mich [\S]*"* (Moby Dick - Melville)
- Zitate/Anlehnungen: Aufschluss auf Textbeziehungen (Intertextualität) und Urheberschaften

Alter

- Chronologische Zuordnung von Phrasen oder markanter Texte
- → Starker Anlehnung lässt auf eher junge Korpora schließen



Struktur

- Verteilung (Häufungen, Abnahme) des Matchingverlaufs in Korpora
- → Aus Struktur lassen sich u.U. thematische Rückschlüsse (Vermutungen) ziehen
- → Grundlage weiterer Nachforschungen

Sprachentwicklung

- Jüngste Sprachtendenzen z.B. auf Grundlage von Chat-/SMS-/email-Korrespondenzen
- → **Satzumstellungen:** *„Ich komme nicht, weil ich hab keine Zeit"*
- → **Artikeleinsparungen:** *„muss erst Computer heil machen"*
- → **Ethnolektale Formen:** *„ich habe fertig"*

6.4 Beispiele

Mustervarianten:

Korpus: Werke von Shakespeare
Suchtext: „Der Himmel steh mir be(i|y)!"
Filter: [^a-zA-ZäüöÜÖÄß\n]
Methode: Präfix/Permutation/
Case-Tolerant/ Tolerance (+1)

Werk	Varianten
Hamlet	„dem Himmel bey"
Julius Ceasar	„steh unverrückt mir bei"
König Johann	„Der Himmel vergebe mir"
	„bey dem Himmel"
König Lear	„Der Himmel sey mir gnädig"
König Richard II	„beschüze mich der Himmel!"
	„beym Himmel"
	„der Himmel mir vergebe"
	„Der Himmel steh mir bey!"
Othello	„der Himmel gebe mir Gnade"
Romeo und Julia	„weh mir, daß der Himmel"

Altersbestimmung:

Korpus: deutsche Briefe und e-mails
Suchtext: Begrüßungsformeln
Filter: [^a-zA-ZäüöÜÖÄß\n]
Methode: Case-Tolerant

Formel	Datum
Lieber Freund	1850-1950
Lieber [\S]*	1980, 1990
Hallo [\S]*	1995
Hallo ertmal	1998
Mahlzeit	1999
Moin	2003
Halli Hallo	2005
Hi	2005

Korpus: Zeitungsartikel
Suchtext: „du bist [A-Z][\S]*"
„wir sind [A-Z][\S]*"

Slogan	Datum
Du bist Deutschland	2005
Du bist Mozart, Bach, ...	2005
Wir sind Papst	2005
Wir sind Weltmeister	1990

6.4 Beispiele

Thematische Häufungen:

Korpus: Klassiker (Gutenberg-Projekt)

Suchtext: „thank (god|goodness)"

Filter: [^a-zA-Z\n]

Methode: Permutation/Case-Tolerant/ Tolerance (+1)

Autor	Titel	#Tokens	Perm.	#Match.	Dauer [s]
L. Tolstoi	War and Peace	568633	-	28	6.67
L. Tolstoi	War and Peace	568633	+	34	7.23
B. Stoker	Dracula	160814	-	17	1.13
B. Stoker	Dracula	160814	+	25	1.25
-	Bibel	792000	-	10	4.77
-	Bibel	792000	+	50	5.67
C. Dickens	Dombey and Son	186701	-	7	2.45
C. Dickens	Dombey and Son	186701	+	7	2.89
C. Bronte	Jane Eyre	186701	-	5	1.50
C. Bronte	Jane Eyre	186701	+	7	1.53
H.G. Wells	Time Machine	34253	-	0	0.28
H.G. Wells	Time Machine	34253	+	0	0.33
W. Whitman	Leaves of Grass	123664	-	0	0.86
W. Whitman	Leaves of Grass	123664	+	0	0.97
W. Irving	The Alhambra	120757	-	0	0.74
W. Irving	The Alhambra	120757	+	0	0.86

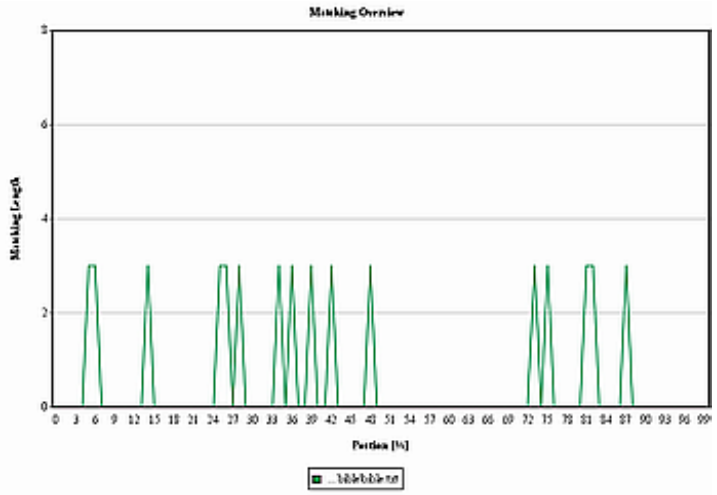
Tokenverteilung:

Korpus: Die Bibel (Altes/Neues Testament)

Suchtext: „(god|lord) said to"

Filter: [0-9]{2}:[0-9]{2}|[^a-zA-Z\n]

Methode: Permutation/Case-Tolerant/ Tolerance (+1)



6.5 Erweiterungsmöglichkeiten

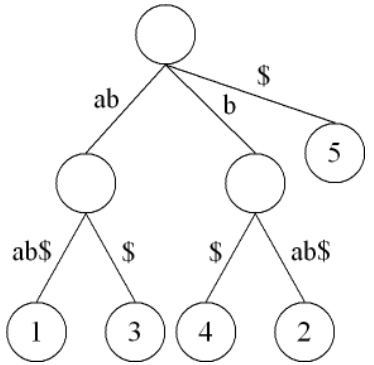
Andere Ähnlichkeitsansätze:

- **Syntaktische Ähnlichkeiten** (Hemmingdistanz)
- **Phonetische Ähnlichkeiten**
  - → Transkription der Token in Lautschrift (z.B. IPA)
  - → Automatismus über Datenbankabfrage
  - → Token-Matching auf Basis von Lautsymbolen

Bsp: *„knight" ['nait] „night" ['nait]*
„two" ['tu:] „to" ['tu:]
„eye" ['ai] „I" ['ai]

Suffix-Bäume:

- Kompakte Darstellung aller (Teil-)Tokenfolgen als Suchbaum (*Bsp: abab$*)
- Pfad von Wurzel zu Blatt repräsentiert eine Teilfolge

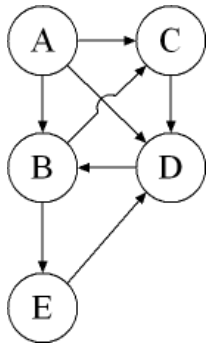


- → **Longest Common Substring**
- → **Longest Repeated Substring**

Adjazenzmatrizen

- Repräsentation des Korpus als Binär-Matrix
- Aufeinanderfolgende Token spannen einen Übergangs-Graphen auf, der als n × n-Matrix dargestellt werden kann

	A	B	C	D	E
A	0	1	1	1	0
B	0	0	1	0	1
C	0	0	0	1	0
D	0	1	0	0	0
E	0	0	0	1	0



- → **Globale Darstellungs-/Vergleichsmöglichkeit von Korpora**