Projektbericht pronto: Probleme der Eigennamenerkennung

Katharina Kober, Alexander Krumeich,
Klaus von der Landwehr, Hagen Langer, Georg Rehm

http://www.cl-ki.uni-osnabrueck.de/pronto/
pronto@cl-ki.uni-osnabrueck.de

Vorläufige Version vom 14. Juni 2002, nur zum internen Gebrauch

Inhaltsverzeichnis

1. Einführung/Zielsetzung

Die korrekte Zuordnung einer Wortform zu einer morphologischen, syntaktischen, semantischen oder lexikalischen Kategorie ist eine zwingende Voraussetzung für fast alle anspruchsvollen Sprachanalyseprozesse. Fehler bzw. Uneindeutigkeiten in einer Analysekomponente, die diese Klassifikationen durchführt, schlagen sich in Fehlern bzw. Performanzverlusten in allen Komponenten nieder, die diese Informationen weiterverarbeiten (Wauschkuhn, 1995). Der Bereich der Eigennamen zählt zu den Wortklassen, bei denen eine solche eindeutige kategoriale Zuordnung in der Regel besonders schwierig ist.

Dies gilt nur eingeschränkt für die Mehrzahl der Sprachen, in denen Eigennamen innerhalb eines Satzes durch Großschreibung markiert sind; für das Englische sind z. B. bereits Erkennungsraten von 97 % Recall und 91 % Precision erzielt worden (Göser, 1997), diese Werte liegen aber (insbesondere hinsichtlich der Precision) immer noch deutlich unter den Erkennungsraten allgemeiner Tagging-Systeme, die inzwischen deutlich über 97 % Erkennungsleistung erbringen.

Ein etwas sophistizierteres Tagging-System stellen Mani und MacMillan (1996) vor. Hier sollen nicht nur Namen in Texten gefunden werden, sondern gleichzeitig auch rudimentäre Diskursstrukturen aufgebaut werden (siehe Abschnitt 5.1).

Im Deutschen ist die Erkennung von Eigennamen vor allem wegen der Großschreibung von Nomina erheblich schwieriger. Zu den weiteren Hauptproblemen zählen:

- morphologische Markierungen – abgesehen vom Genitiv-s – fehlen systematisch
- bei Eigennamen ist eine besonders hohe Anzahl von Homonymien mit Elementen gerade jener Klasse zu beobachten, von der sich die Eigennamen wegen zusätzlicher syntaktischer, morphologischer und orthographischer Parallelen so schwer abgrenzen lassen, nämlich den Nomina, insbesondere in den folgenden Bereichen:
  - Berufsbezeichnungen: *Richter*, *Bauer*, *Kaufmann* etc.
  - Jahreszeiten: *Sommer*, *Winter*
  - Tier- und Pflanzennamen: *Hase*, *Baum*, *Blume* usw.
  - Bezeichnungen fuer Orte und Regionen, bzw. daraus abgeleitete Substantive: *Bayer*, *Sachs(e)*, *Holländer* ...
- morphophonologisch bzw. morphographematisch unterliegen Eigennamen praktisch keinen systematischen Restriktionen
- Steiner (1995) weist darauf hin, daß Namen auch als Mehrwortlexeme auftreten können, die dann als eigene Nominalphrase behandelt werden.

Das hier vorgestellte System `pronto` (*Proper noun tagger Osnabrück*) entstand im Rahmen des Hauptseminars „Korpuslinguistik“ im Studiengang „Allgemeine Sprachwissenschaft“ an der Universität Osnabrück im Wintersemester 1997/98. Ausgangspunkt des Seminars waren Entwicklungen, die in den frühen 90er Jahren im Bereich *Very Large Corpora* (VLC) gemacht wurden (Church und Mercer, 1993). In Anlehnung an die Arbeit von Gotthard und Seiffert (1997) entschieden wir uns, ein System zu entwickeln, das in der Lage ist, Personennamen in deutschsprachigen Texten zu markieren.

2. Ein naiver Ansatz

Im ersten Stadium des Projekts bestand unsere Herangehensweise darin, typische Suffixe von Eigennamen auf reguläre Ausdrücke abzubilden. Diese regulären Ausdrücke formen den Kern eines `flex`-Programms[1] (Levine et al., 1992; Herold, 1995), das wir als Filter für Eigennamen verwenden wollten. Unsere initialen Suffixregeln (für Nachnamen) lauteten

1. `{Wortanfang}(v|w)sk(i|ij|y){Wortende}`
2. `{Wortanfang}(us|a){Wortende}`
3. `{Wortanfang}er{Wortende}`
4. `{Wortanfang}man(n){Wortende}`

wobei `Wortanfang` eine beliebige Buchstabenkette ist, die mit einem Großbuchstaben beginnt (`[A-Z][a-z]*`) und beliebiger Zwischenraum (sog. *white space*) das `Wortende` markiert (`(|\t|\n)`).

Regel 1 trifft auf die slawischen Namen, wie sie z. B. im Ruhrgebiet geläufig sind (z. B. Lewandowsky oder Katlewski). Regel 2 erreicht alle Nachnamen auf *-us* oder *-a*. Berufsbezeichnungen wie Bäcker, Richter oder Schlosser, die häufig als Namen auftauchen, sind die Domäne der Regel 3. Regel 4 feuert bei Namen wie Hofman, Lehmann etc. Besonders aufgrund der Regeln 2 und 3 erreichten wir bei ersten Versuchen unerwartet hohe Recall-Werte, die jedoch in großer Diskrepanz zur Precision standen. Worte, die auf *-er* oder *-a* enden, gibt es reichlich und die wenigsten davon sind zweifelsfrei Namen. Getestet wurden die Regeln in diesem Projektstadium mittels eines Auszugs aus Goethes „Italienischer Reise“, den wir zum Vergleich von Hand annotiert hatten. Bei Vornamen ergaben sich ähnliche Ergebnisse. Um dieses Manko auszubessern, versuchten wir, aus größeren Namenslisten (weiterhin manuell) reguläre Ausdrücke zu gewinnen, die genauere Trefferquoten für Namen liefern sollten.

Dazu verwendeten wir verschiedene Arten von Namenslexika. Zuerst diente uns eine Liste von ca. 1400 Autoren, die Newsartikel in der de-Hierarchie des Usenet geschrieben haben. Die Vor- und Nachnamen lassen sich aus diesen Artikeln relativ leicht extrahieren, da sie im reservierten Bereich (dem sog. *header*) der Nachricht stehen und weiterhin der Kette `From:` folgen. Das Lexikon war sehr variantenreich, enthielt aber eine nicht unerhebliche Menge unbrauchbare Daten, da die Teilnehmer des Usenet teilweise nicht ihre echten Namen, sondern Pseudonyme verwenden.

Weniger variantenreich, dafür aber wesentlich weniger ‘verrauscht’ ist die Benutzerliste des Mailservers des Rechenzentrums der Universität Osnabrück, die wir als zweites Lexikon zur Verfügung hatten. Diese beiden Lexika waren ausreichend klein, um eine manuelle Bearbeitung zu ermöglichen.

Bereits früh erkannten wir, daß Eigennamen sehr komplex aufgebaut sein können. Beispielsweise existieren eine Vielzahl von Titeln, die als Bestandteile eines Namens mit erkannt werden sollen. Dazu gehören sowohl akademische als auch Adelstitel. Wir mußten

[1] `flex` ist die `lex` Implementierung aus der GNU-Programmsammlung (vgl. Levine et al., 1992, S. 340)

somit herausfinden, ob es eine kanonische Struktur dieser Titel gibt (siehe Abschnitt 3.1). Ebenfalls problematisch sind Ambiguitäten von Namen. Diese lassen sich in unterschiedliche Klassen einordnen: Zum einen Berufsbezeichnungen, die auch als Namen auftreten können (z. B. Bäcker, Richter, Bauer), zum anderen Homonyme wie August oder York[2].

Ein weiteres Problem bestand in der Verfügbarkeit von Daten. Die „Italienische Reise“, die uns zu Beginn als Testmenge gedient hatte, war zu klein und nicht variantenreich genug, um mit der Entwicklung unserer Namenslexika standzuhalten. Zudem sind die Namen, die in diesem Text auftreten z. T. anachronistisch, so daß die Trefferraten allein an der Diachronizität des verwendeten Materials leiden. Es mußte also ein geeignetes Korpus herangezogen werden. Wir entschieden uns für ein Mischkorpus, das größtenteils aus Artikeln aus der Süddeutschen Zeitung besteht. Teile dieses Korpus wurden von uns manuell annotiert, so daß wir einen Referenztext haben, an dem wir Recall und Precision des Systems ablesen können.

3. Die linguistische Ebene

3.1. Komplexe Namen als reguläre Ausdrücke

Die möglichen Bestandteile eines komplexen Eigennamens seien folgende (vollständige Listen für verschiedene der u.g. Teilbereiche befinden sich in Anhang A. Die mit * markierten Bestandteile sind bisher noch nicht implementiert):

- Anrede (Herr, Frau, Geschwister ...)
- Adelstitel (Freiherr, Freifrau, Baron, Baronin ...)
- klerikaler Titel (Seine Eminenz, Seine Heiligkeit ...) *
- militärischer Rang (Oberst, Admiral ...) *
- Akademischer Titel (Apl., Prof., Dr. (h. c.) ...)
- Vorname(n)
- Adelspräposition (von, vom und zum, van ...)
- Nachname(n)
- Generationsmarke (jun., sen.)
- Beiname (die Große, der Seefahrer ...) *

Alle Bestandteile eines komplexen Eigennamens, die weder Vor- noch Nachname sind, können durch relativ kurze aber nahezu vollständige Listen erfasst werden. Je nach Kontext können mehrere Bestandteile auch einfach wegfallen. Aufgrund der strengen formalen Struktur von komplexen Eigennamen kann das Vorkommen von mehreren Bestandteilen

[2] Ein Teil des Namens der Stadt New York wird als der Vorname *York* erkannt.

als sicheres Anzeichen für einen Namen angesehen werden. Allerdings sind die meisten vorkommenden Namen nicht komplex.

Ein Ausdruck in `flex`, der die formale Struktur von komplexen Eigennamen ausnutzt, besteht zunächst aus den oben aufgeführten Listen von Namensbestandteilen, die in der entsprechenden Reihenfolge in den Ausdruck integriert werden. Dies ergibt ein Makrosuchmuster, welches nur eine einzige Möglichkeit der Abfolge aller Bestandteile zuläßt. Intern können die einzelnen Listen als einfache Mengen von Alternativen repräsentiert sein. Allerdings ähneln sich sehr viele Listeneinträge, so daß sich hier Mikrosuchmuster erzeugen lassen, die den Suchraum begrenzen.

3.2. Berufsbezeichnungen

Berufsbezeichnungen sind bei der Eigennamenerkennung nicht ohne Bedeutung. Besonders häufig tritt die Kombination von Berufsbezeichnungen und Eigennamen in Zeitungsartikeln auf, da Personen hier im Zusammenhang mit ihrem Beruf vorgestellt werden: „Die Maiwoche wird Freitag (15.5.) durch *Oberbürgermeister Hans-Jürgen Fip* offiziell eröffnet." [aus einer Osnabrücker Wochenzeitung vom 13.05.98] Wird eine Berufsbezeichnung vor einem String als solche erkannt, dann erhöht dies die Wahrscheinlichkeit, daß der folgende String tatsächlich ein Eigenname ist. Die Erkennung der Berufsbezeichnungen wird durch eine Komponente geleistet, die diese als reguläre Ausdrücke betrachtet. Diese Repräsentation ermöglicht eine einfache Erfassung und Pflege von flektierten Formen und eine effiziente Erkennung im Text. Die Berufsbezeichnungen sind in Gruppen zusammengefasst, von denen die meisten Berufe enthalten, die auf -er enden. Eine weitere Gruppe beinhaltet Berufsbezeichnungen, die aus dem Französischen stammen, wie Friseur und Ingenieur und die letzte solche Berufe, die weder auf -er enden noch aus dem Französischen stammen, wie z.B. Arzt (siehe unten `BERUFREST`).

(1)
```
AMTBERUFNOUN (Autohersteller|Autoknacker|Babysitter|Bademeister|Baecker|
Bandleader|Bastler|Bauarbeiter|Bauer|Bauleiter|Baumeister|[A-Z][a-z]+min
ister|Bauunternehmer|Bauzeichner|Bearbeiter|Befehlshaber|Bereichsleiter|
Bergarbeiter|Berichterstatter|Berufsberater|Berufsfahrer|Berufspendler|B
erufspolitiker|Berufsschullehrer|Betonbauer|Betriebsberater|Betriebsinha
ber|Betriebsleiter|Betriebswirtschaftler|Bewacher|Bierbrauer|Bildreporte
r|Bildungsminister|Bildungspolitiker|Binder|Binnenschiffer|Boettcher|Bot
aniker|Botschafter|Boxer|Boxweltmeister|Braumeister|Buchbinder|Buchhalte
r|Buergermeister|Buergerrechtler|Bundesforschungsminister|Bundesgesundhe
itsminister|Bundesjustizminister|Bundesminister|Bundessieger|Bundesverfa
ssungsrichter|Bundeswirtschaftsminister|Busfahrer|Charakterdarsteller|Ch
efberater|Cheftrainer|Chemiker|Computerhersteller|Controller|Darsteller|
Datenbankbetreiber|Dealer|Dekorationsmaler|Denkmalpfleger|[A-Z][a-z]+leh
rer|Devisenbringer|Dichter|Diener|Dienstanbieter|Dramatiker|Drehbuchschr
eiber|Dreher|Einbrecher|Einsatzleiter|Elektriker|Elektrotechniker|Energi
eminister|Entertainer|Entscheider|Entwickler)
```

Für die Berufsbezeichnungen auf -er sind zudem Regeln angegeben, in welcher Weise sie flektiert werden können. Da sich dies bei der Gruppe `BERUFREST` nicht allgemein angeben lässt, sind die Flektionsmorpheme hier mit Hilfe von regulären Ausdrücken für

jede einzelne Bezeichnung explizit angegeben. Dieser Repräsentationsmechanismus ermöglicht es zudem, verschiedene Varianten einer Berufsbezeichnung zusammenzufassen. Statt Finanzexperte, Wirtschaftsexperte, Computerexperte u.s.w. einzeln aufzuzählen, wird einfach: `[A-Z][a-z]+experte(n)` angegeben.

(2)
```
BERUFREST (Abgeordente(r|n)?|Architekt(s|en|in)?|Artist(en|in)?|[A-Z][a-
z]+arzt(es)?|Arzt|Aerztin|Bedienung|[A-Z][a-z]+chef(s|in)?|Chef|Chirurg(
s|en|in)?|Dentist(en|in)?|Direktor(s|in)?|Dozent(en|in)?|[A-Z][a-z]+expe
rte(n)?|Experte(n)?|Expertin|Fotograf(s|en|in)?|Galerist(s|en|in)?|Hebam
me|[A-Z][a-z]+intendant(en|in)?|Intendant|Kaufmann(s)?|Kauffrau|Kapitaen
(s|in)?|Koch(s)?|Koechin|Landrat(s)|Landraetin|Linguist(en|in)?|Matrose(
n)?|Matrosin|[A-Z][a-z]+mitglied(s)?|Mitglied|[A-Z][a-z]+ologe(n)?|[A-Z]
[a-z]+ologin|[A-Z][a-z]+praesident(en|in)?|Praesident|Putzfrau|Putzmann|
Seemann|Servicekraft|[A-Z][a-z]+sprecher(s|in)?|Sprecher|Stadtrat(s)?|St
adtraetin|Student(en|in)?|Studierende(r|n)?|[A-Z][a-z]+vorsitzende(r|n)?
|Vorsitzende(r|n)?|[A-Z][a-z]+wirt(s|in)?|Wirt|Zimmermann(s)?)
```

Berufsbezeichnungen haben aber nicht nur positive Effekte auf die Eigennamenerkennung. Probleme machen ganz besonders solche Fälle: „Die Angeklagten hätten Angst und Schrecken verbreitet und seien mit Drohungen und Gewalt schnell bei der Hand gewesen, sagte der Vorsitzende Richter in der Urteilsbegründung.“ [Testkorpus] Der String „Richter“ kann in diesem Fall sowohl die Berufsbezeichnung als auch der Nachname einer Person sein. Einem menschlichen Leser fällt die Disambiguierung in diesem Beispiel aufgrund des im gleichen Satz gegebenen Kontextes (wie „Angeklagte“ und „Urteilsbegründung“) nicht allzu schwer und wählt so die Berufsbezeichnung. Wäre der Kontext nicht so eindeutig, hätte auch ein Mensch Probleme, die intendierte Lesart zu erkennen. Wesentlich leichter zu verstehen sind solche Konstruktionen, in denen die mögliche Berufsbezeichnung tatsächlich der Nachname[3] ist, wie in den folgenden Beispielen:

1. „Eduard *Zimmermann* gibt Entwarnung.“
2. „Manfred *Kaufmann* Direktor der Gynäkologischen Klinik der Universität Frankfurt warnt jedoch vor einer Verunsicherung der Frauen.“
3. „. . . seine Leute hätten soeben Jürgen *Schneider* aufgespürt.“
4. „Diese Klavierlieder wurden durch den Regisseur Ernst Theo *Richter* zu einem sensiblen und intelligenten Kammerspiel.“

Auch `pronto` kann mit diesen Fällen problemlos umgehen. Die Disambiguierung kann durch die anderen gefundenen Namensbestandteile in unmittelbarer Umgebung des zweideutigen Strings wie folgt vonstatten gehen: In den oben angegebenen Beispielen wurde immer der volle Name verwendet. Der zweideutige String steht also hinter einem (auch

[3] Crystal (1995, S. 112) führt in diesem Zusammenhang *Schmidt* als Beispiel für einen Nachnamen an, der in vielen Regionen Europas und darüber hinaus gebräuchlich ist: arabisch *Haddard*, ungarisch *Kovács*, russisch *Kusnetzow*, portugiesisch *Ferreiro*, englisch *Smith*, spanisch *Hernández/Fernández*, französisch *Le Fèvre/La Forge* etc.

als solchen erkannten) Vornamen. Berufsbezeichnungen stehen aber in der Regel nicht hinter einem möglichen Vornamen, sondern davor. Würde statt des Vornamens eine Anrede wie „Herr“ oder „Frau“ stehen, verringert dies ebenfalls die Wahrscheinlichkeit, daß es sich bei dem nachfolgenden String um eine Berufsbezeichnung handelt, da diese normalerweise nicht mit Anreden kombiniert werden (siehe auch Abschnitt 3.1). Potentielle Berufsbezeichnungen können `pronto` zwar in gewissen Fällen zum Problem werden, aber erkannte Berufbezeichnungen stellen einen wertvollen Hinweis für die Erkennung des nachfolgenden Eigennamens dar.

4. Implementierung

Angelehnt an die übliche Methodik im Bereich VLC entwickelten wir zunächst ein Trigramm-Modell einer Namensliste, das sich jedoch als nicht tauglich erwies, da Namen strukturell zu große Entropie aufweisen.

`pronto` sollte mit möglichst großer Flexibilität ausgestattet sein. Daher soll es nicht als ein opakes System, sondern für den Anwender transparent als eine Kaskade kleinerer Programme arbeiten. `pronto` wird unter UNIX (primär unter LINUX 2.0.33 und SOLARIS 2.6) vor allem mit Werkzeugen aus der GNU-Programmsammlung[4] entwickelt. Die Architektur mehrerer zu verkettender Module entspricht einem essentiellen Arbeitsparadigma der UNIX Umgebung, durch Kombination von Programmen neue Funktionalität zu gewinnen (vgl. Kernighan und Pike, 1986; Gancarz, 1995) und nutzt die dort vorhandene Funktionalität (Ein-Ausgabeumleitung, Pipes, Shell-Skripte) aus.

Die einzelnen Module arbeiten als Filter über einem Eingabetext. Jede einzelne Komponente liest den kompletten Text ein und berechnet Hypothesen. Diese Hypothesen werden im Text vermerkt. Dazu überlegten wir uns eine Notation, die von allen Programmen verwendet wird (siehe Abschnitt 4.3). Als Schnittstelle zwischen den einzelnen Komponenten, die sequentiell als eigene Prozesse laufen, fungiert eine UNIX Pipe.

4.1. Lexikalische Analyse mittels `flex`

Die erste Herangehensweise war, Namen auf typische Muster bei Suffixen hin zu untersuchen und diese Muster als reguläre Ausdrücke zu formulieren, siehe Abschnitt 2. Diese bildeten die Grundlage eines `flex` (vgl. Levine et al., 1992; Herold, 1995) Programms, das den zu untersuchenden Text auf Namensmuster durchsucht. Für dieses Programm wurden die Einträge im Namenslexikon zu einem großen regulären Ausdruck konkateniert, der in etwa die Form `Aaron|Albert|...|Zinnober` hatte.

In dem Maße, wie unsere Namenslexika an Umfang gewannen (siehe Abschnitt 4.2), stellte sich heraus, daß `flex` für unsere Zwecke keinen ausreichend effizienten Zugriff auf die Lexikondaten bietet. Erste Probleme mit dem Überlauf interner Speicher konnten wir noch durch Änderung des `flex`-Quellcodes und anschließende Neukompilierung des

[4] Siehe `http://www.gnu.org`

Quelle		Anzahl Namen
D-Info-2	Augsburg	148.577
	Bochum	151.071
	Braunschweig	129.272
	Bremen	291.560
	Cottbus	52.084
	Freiburg	107.183
	Kiel	146.714
	Mainz	94.220
	Osnabrück	79.571
	Ulm	52.306
	Σ	1.252.558
UB-Liste		308.755
PND		10.324.317

Tabelle 1: Die lexikalischen Rohdaten

Programms lindern[5], doch auch danach ließ die Performanz noch sehr zu wünschen übrig. Als Schritt in die richtige Richtung erwies es sich, die Daten statt in einer „flachen“ `flex` Quelldatei in einem Trie abzulegen, da diese Datenstruktur einen wesentlich schnelleren Zugriff ermöglicht. Der hier verwendete Trie fand bereits erfolgreich im GEPARD System Verwendung.

4.2. Konstruktion der Lexika

Zur Gewinnung von möglichst umfangreichen Namenslisten stand uns die D-Info-2 CD Rom mit den deutschen Telefonbüchern zur Verfügung D-Info 2 (1995). Da uns ein Arbeiten auf den Gesamtdaten zu umfangreich erschien, haben wir zehn regional möglichst gleichmäßig verteilte Städte (vgl. Tabellen 1 und 2) ausgewählt und die Telefonbücher dieser Städte über die Export-Funktion des Datenbank-Frontends der CD Rom in einem ASCII-Format extrahiert, wobei insgesamt ca. 83 Megabyte Daten entstanden sind. Die folgende Liste (3) zeigt einen Auszug aus dem Augsburger Telefonbuch, wobei deutlich wird, daß eine Extraktion der Vor- und Nachnamen nicht ohne weiteres möglich war.

[5] Bei der Verarbeitung von Lexika mit mehr als ca. 3000-4000 Einträgen traten solche Überläufe auf. Vern Paxson, der Autor von `flex`, gab uns folgende Hinweise auf zu ändernde Definitionen in der Datei `flexdef.h` (in Klammern jeweils der Originalwert der `flex` Programmquellen):

```
#define INITIAL_MNS          64000     (2000)
#define MNS_INCREMENT        1024000   (1000)
#define INITIAL_MAX_DFAS     3000      (1000)
#define MAX_DFAS_INCREMENT   3000      (1000)
#define JAMSTATE             -655320   (-32766)
#define MAXIMUM_MNS          64000     (32000)
```

Quelle		Anzahl Vornamen	Anzahl Nachnamen
D-Info-2	Augsburg	6.784	32.264
	Bochum	5.782	36.235
	Braunschweig	5.563	30.419
	Bremen	10.051	51.444
	Cottbus	2.309	12.765
	Freiburg	5.064	24.526
	Kiel	5.442	28.684
	Mainz	5.302	24.368
	Osnabrück	4.302	20.996
	Ulm	3.915	15.286
	Σ	25.069	148.361
UB-Liste		16.549	114.982
PND		21.467	58.482
UB-Liste $\cap$ D-Info-2		5.939	41.281
UB-Liste $\cap$ D-Info-2 $\cap$ PND		2.225	13.566

Tabelle 2: Die extrahierten lexikalischen Daten (Doubletten entfernt)

(3)
```
"3A Wassertechnik";"Argonstr. 7";"86153 Augsburg";"0821-5584563"
"A A B Treuhand GmbH Steuerberatungsgesellschaft";"Brunnenbachstr. [...]
"A-AS Auto-Service GmbH AutoRepar.";"Schiessgrabenstr. 6";"86150 [...]
"A Be Fi S Aerzte-Beratungs- und Finanzierungs-Service GmbH";"[...]
"";"";"861.. Augsburg";"0821-718959"
[...]
"Adametz Erna";"Vord. Lech 49";"861.. Augsburg";"0821-518199"
"Adametz Manfred";"Werderstr. 18";"86159 Augsburg";"0821-579176"
"Adamidov Eleni";"";"861.. Augsburg";"0821-95598"
"Adamietz Andreas";"Bluecherstr. 59";"86165 Augsburg";"0821-716492"
"Adamietz Christian";"Seilerstr. 4";"86153 Augsburg";"0821-553808"
"Adamietz Gregor";"H
```

Mit Hilfe bekannter Werkzeuge aus der GNU-Programmsammlung (z. B. `recode`, `comm`, `cut`, `wc`, `grep`, `bash`) haben wir daraufhin ein Shell-Skript erstellt, das automatisch die Vor- und Nachnamen aus den Rohdaten herausfiltert. Die Prämisse für diese Vorgehensweise ist, daß die von uns gewünschten Token in den vorliegenden Listen in der From `"Vorname Nachname"` enthalten sind, wobei jedoch ein Zugriff auf die erste bzw. zweite Spalte aus den Dateien nicht direkt möglich war, wie der erste Teil in (3) zeigt. Daher wurden zunächst alle Zeilen, die mit einer Zahl beginnen oder in denen in einem Wort mehr als ein Großbuchstabe auftaucht (ein Hinweis auf einen Firmennamen oder eine Abkürzung) getilgt. Selbiges geschah mit allen Zeilen, in denen explizit `mbH`, `e.V.`, `mbR`, `Gesellschaft` o. ä. vorkommt. Über weitere reguläre Ausdrücke wurden Zeilen, die in den Namensfeldern Abkürzungen enthalten, ebenfalls aus dem Korpus getilgt. Nach einigen

Umformatierungen, wobei auch Ketten, die lediglich aus einem oder zwei Zeichen bestanden, getilgt wurden, lagen schließlich die Vor- und Nachnamen in getrennten Dateien vor, durchsetzt von vielen unerwünschten Lexemen wie `Altenheim` oder `Spielplatz`, die unser Konvertierungsprogramm nicht korrekt erkannt hatte. Erste Tests mit einem Tagger, der Namen ausschließlich anhand der aus der CD Rom gewonnenen Daten markierte, waren sehr enttäuschend, da der Anteil 'herkömmlicher' Nomina in den Wortlisten höher war, als wir befürchtet hatten.

Zur Erweiterung unseres Datenbestandes stellte uns daraufhin die Bibliothek der Universität Osnabrück eine Datei (in Tabelle 1 und 2 als UB-Liste bezeichnet) mit den Namen der im lokalen OPAC System erfaßten Autoren zur Verfügung[6]. Diese Datei wurde ebenfalls durch ein Shell Skript konvertiert, so daß sich zwei getrennte Listen mit ca. 16.000 Vor- und ca. 114.000 Nachnamen ergaben, vgl. Tabelle 2. Problematisch war im Zusammenhang der Extraktion der Namen vor allem die inkonsistente Erfassung von Namenszusätzen. Der Name *Alice ter Meulen* wurde beispielsweise einerseits als *Meulen, Alice ter* aber auch als *TerMeulen, Alice* erfaßt, wodurch unbeabsichtigt nicht korrekte Daten (*TerMeulen*) entstanden sind. Weiterhin existiert zwischen den beiden genannten Einträgen keine Verbindung, so daß ein Datenbank-Frontend nicht zwischen Schriften von *Meulen, Alice ter* und *TerMeulen, Alice* unterscheiden kann. Lösungen für dieses Problem wurden im Rahmen des Projekts OSIRIS[7] entwickelt (vgl. beispielsweise Ronthaler und Sauer, 1997).

Eine erneute Erweiterung unseres Datenbestandes nahmen wir mit Hilfe der Personennamendatei (PND) der Deutschen Bibliothek vor (vgl. PND, 1996). Diese Datei wird im Rahmen eines von der DFG geförderten Projekts entwickelt, um „normierte Personennamen" bereitzustellen, „die für die aktuelle Formal- und Sacherschließung, die Altbestandserschließung sowie nationale Katalogisierungsprojekte (z. B. Autographen- und Nachlaßerschließung [...]) wesentlich sind, jedoch ohne Vollständigkeit anzustreben" (PND, 1996, S. 7). Es folgen einige Beispieleinträge:

(4)
```
816 Lachner, Franz: Alidia. - [1838]
816 Bauhuis, Bernard: Epigrammata. - 1618
816 Hieronymus, Sophronius E.: Divina Bibliotheca
816 Anania, Giovanni L.: De natura daemonum libri IV. - 1581
816 Andreae, Antonius: Scriptum in arte veteri. - 1508
```

Aufgrund der sehr konsistenten Notationsweise der Daten (nach einer mehrstelligen Zahl zu Beginn der Zeile folgt der Nachname, getrennt durch ein Komma folgen Vornamen bis zu einem Doppelpunkt) konnten die für uns relevanten Felder wiederum mit Hilfe eines Shell Skripts extrahiert werden.

[6] An dieser Stelle möchten wir uns bei Hartmut Zillmann, Universitätsbibliothek Osnabrück, und Marc Ronthaler, Institut für Semantische Informationsverarbeitung, Universität Osnabrück, für die Zurverfügungstellung der Daten bedanken.

[7] Osnabrück Intelligent Research Information System – Ein Gemeinschaftsprojekt der Universitätsbibliothek Osnabrück und des Instituts für Semantische Informationsverarbeitung der Universität Osnabrück, gefördert von der Deutschen Forschungsgemeinschaft (DFG), zu erreichen unter http://osiris.ub.uni-osnabrueck.de/db/start.html

Nachdem wir nun aus den drei Quellen (D-Info-2, PND, UB-Liste) jeweils zwei Listen mit Vor- bzw. Nachnamen gewonnen hatten, haben wir mit dem GNU Werkzeug `comm` die Listen in zwei Stufen miteinander abgeglichen, d.h. Einträge, die sowohl in der Liste der Bibliothek als auch in den D-Info-2 Daten vorliegen, wurden in eine dritte Datei geschrieben. Es ist kaum anzunehmen, daß in beiden Listen die gleichen fehlerhaften Einträge (z. B. das bereits erwähnte `Altenheim`) enthalten sind, so daß man davon ausgehen kann, daß die durch diesen Abgleich generierte Liste mit einer sehr hohen Wahrscheinlichkeit ausschließlich Namen enthält. Die auf diese Weise gewonnenen Vor- und Nachnamenlisten wurden dann erneut mit den aus der PND-Datei generierten Listen abgeglichen, die ihrerseits schon kaum fehlerhafte Einträge aufwiesen. So wurden Listen erstellt, die 2.225 Vor- und 13.566 Nachnamen enthalten, siehe auch Tabelle 2.

4.3. Annotationsschema

Das `pronto` System besteht aus verschiedenen Modulen (siehe Abschnitt **??**), die Teile eines Datenstroms modifizieren. Konkret bedeutet dies, daß das erste Modul (z. B. `nounscan`) einen ASCII Text einliest, gegebenenfalls Worte annotiert und dann den modifizierten Text in seiner Gesamtheit in einer UNIX Pipe an das nächste Modul weiterreicht. Die Komponente `heval` bildet im Regelfall die letzte Komponente einer solchen Kette. Wichtig ist, daß keines der Module Teile des Eingabetextes tilgen kann.

Aufgrund dieser Architektur muß gewährleistet sein, daß verschiedene Komponenten ein und dasselbe Wort annotieren können. Aus diesem Grund haben wir uns entschieden, Markierungen durch das `pronto` System durch zwei Doppelkreuze (`##`) einzuleiten und zu beenden. Der so isolierte Text besteht aus dem erkannten Wort, gefolgt von einem Pipe-Zeichen (|). Zwischen dem Pipe-Zeichen und den schließenden Doppelkreuzen tragen die Komponenten zunächst ihren eigenen Namen (also z. B. `nachpnd.l`) ein, gefolgt von der Art der erkannten Kette.[8] Den letzten Bestandteil dieses Annotations-Tripels bildet eine Wahrscheinlichkeit. Es folgen einige Beispiele, wobei darauf verzichtet wird, den Kontext der jeweiligen Namen mit aufzunehmen:

```
##David|nachpnd.l NACHNAME 1.0 vorall.l VORNAME 1.0
     nounscan.l NAME 0.1 ##
##Ho|nachpnd.l NACHNAME 1.0 stopwort.l NAME 0.1 ##
##Zweifel|nachpnd.l NACHNAME 1.0##
##Zeit|nachpnd.l NACHNAME 1.0 nounscan.l NAME 0.1 ##
```
(5)

Erkennt ein Modul ein Wort, so wird zunächst überprüft, ob schon andere Komponenten Annotationen an diesem Wort vorgenommen haben. Ist dies der Fall, so fügt das Modul lediglich ein Tripel der oben beschriebenen Art in die durch die beiden Doppelkreuze abgetrennte Umgebung ein. Wenn nicht, erzeugt das Modul diese Umgebung. Die Komponente `heval` evaluiert als letzter Bestandteil der Modul-Kette die verschiedenen Hypothesen, tilgt sie aus dem Eingabestrom und gibt eine Statistik aus.

[8] Zum gegenwärtigen Zeitpunkt erkennt `pronto` die Kategorien `VORNAME`, `NACHNAME`, `DET`, `ADJEKTIV` und `SONSTIGE`

4.3.1. Das Tag-Set zur Behandlung von Namen und Daten aus den TEI-Richtlinien als Alternative?

Unter der Kennziffer 8879 wurde 1986 die *Standard Generalized Markup Language* (SGML, vgl. Goldfarb, 1990) von der ISO als internationale Norm verabschiedet. SGML legt eine Metasprache fest, die es ermöglicht, eine unendlich große Anzahl von Auszeichnungssprachen zu definieren, von denen die *Hypertext Markup Language* (HTML) die wohl bekannteste Instanz ist. In der Computerlinguistik wird SGML in den verschiedensten Bereichen eingesetzt: So berichtet beispielsweise Volk (1998) von einer Testsatz-Sammlung, die mit Hilfe von SGML annotiert wurde, was Plattformunabhängigkeit garantiert und somit den Austausch und die Erweiterung der Daten vereinfacht. Weiterhin ist eine Validierung der Datenbasis mit einem SGML Parser und der zugrundeliegenden *Document Type Definition* (DTD, eine kontextfreie Syntax, die u.a. die Verschachtelung der Tags definiert) möglich. Die Language Technology Group der Universität von Edinburgh arbeitet seit einigen Jahren an verschiedenen SGML-basierten Werkzeugen, z. B. einem Wortarten-Tagger oder einer C-basierten Programmierschnittstelle (vgl. McKelvie et al., 1997), um sehr große Korpora zu annotieren und zu analysieren. Mit Hilfe des auf der Text Encoding Initiative (TEI) basierenden *Corpus Document Interchange Format* (CDIF) wurde das *British National Corpus* (BNC) annotiert (vgl. Leech et al., 1994).

1987 wurde die Text Encoding Initiative (TEI) von der Association for Computers and the Humanities (ACH), der Association for Computational Linguistics (ACL) und der Association for Literary and Linguistic Computing (ALLC) ins Leben gerufen, um einen einheitlichen Standard zur Auszeichnung von elektronisch verfügbaren Texten zu entwickeln (vgl. für eine knappe Einführung Ide, 1994). Neben Tag-Sets zur Annotation von Dramen, Gedichten, Prosa oder zur Transskription gesprochener Sprache befindet sich in der aktuellen Version P3 der TEI Richtlinien auch ein Tag-Set, um Namen in beliebigen Texten zu markieren. Dieses Tag-Set (Sperberg-McQueen und Burnard, 1994, S. 583 ff.) ist kein *core* oder *base tag set*, sondern kann als *additional tag set* bei Bedarf hinzugezogen werden, um die unterschiedlichen Bestandteile eines Namens wie beispielsweise „Franklin D. Roosevelt“ zu kennzeichnen:

(6)
```
<persName key=FDR1>
  <surname>Roosevelt</surname>
  <forename>Franklin</forename>
  <forename>Delano</forename>
</persName>
```

Der Personenname (markiert mit dem Tag `<persName>`) verfügt in Beispiel (6) über einen eindeutigen Schlüssel – realisiert durch das Attribut `key` –, mit dem man sich auf diesen Namen und die in ihm enthaltenen Elemente beziehen kann. Das SGML-Element `<surname>` umschließt den Nachnamen, entsprechend kennzeichnen zwei Vorkommen des Elementes `<forename>` die Vornamen. Etwas komplexer ist das folgende Beispiel (7):

Tag	Kennzeichnet
`<persName>`	Personennamen in Form eines Nomens oder einer komplexen NP
`<surname>`	Familiennamen
`<forename>`	Vornamen
`<roleName>`	Komponenten, die auf eine gewisse gesellschaftliche Stellung oder einen Rang hinweisen
`<addName>`	Spitznamen, Künstlernamen etc.
`<nameLink>`	Konnektoren wie z. B. „van der“
`<genName>`	Komponenten wie beispielsweise „Junior“

Tabelle 3: Von der TEI vorgeschlagene Tags zur Annotation von Personennamen

(7)
```
<persName key=EGBR1>
  <roleName type=office>Governor</roleName>
  <forename sort=2>Edmund</forename>
  <forename sort=3 full=init reg='Gerald'>G</forename>.
  <addName type=epithet>Jerry</addName>
  <addName type=epithet>Moonbeam</addName>
  <surname sort=1>Brown</surname>
  <relationship full=abb>Jr</relationship>.
</persName>
```

Der Name „Governor Edmund G. ‘Jerry Moonbeam’ Brown Jr.“ besteht aus verschiedenen Teilen, die in unterschiedlicher Weise markiert werden. Der Titel „Governor“ wird vom Element `<roleName>` umschlossen, da es sich um einen offiziellen Titel handelt, der unabhängig vom Träger existiert. Der im Text lediglich durch eine Majuskel abgekürzte zweite Vorname „Gerald“ wird im Attribut `reg` expandiert. Durch den Wert des Attributs `full` wird angezeigt, daß es sich lediglich um ein Initial handelt. Über das `sort`-Attribut der Elemente `<forename>` und `<surname>` kann eine standardisierte Schreibweise von Namen (z. B. zur Sortierung von großen Namensbeständen nach dem Muster „Brown, Edmund G.“) realisiert werden. Das Element `<addName>` markiert Bestandteile komplexer Namensausdrücke, die nicht mit anderen Tags (vgl. Tabelle 3) annotiert werden können, z. B. kulturspezifische Unterscheidungen wie Beinamen, die durch das Attribut `type=epithet` kenntlich gemacht werden.

Sowohl in der Definition als auch in der Motivation dieses Tag-Sets zur Behandlung von Namen und Daten fallen verschiedene Probleme ins Auge. So ist beispielsweise unklar, weshalb zwar der Tag `<genName>` definiert, aber nicht im Beispiel (7) benutzt wird, wo das Element `<relationship>` zum Einsatz kommt. Auch die Benennung des Attributs `full`, das die Werte `yes`, `abb` oder `init` annehmen kann, ist nicht gerade intuitiv, sinnvoller wäre z. B. der Name `form` gewesen. Problematisch ist weiterhin die Tatsache, daß keine Möglichkeit zur Sortierung bzw. Unterscheidung von Beinamen, die durch `<roleName>`- oder `<addName>`-Elemente markiert werden, in ‘Vor’- und ‘Nachnamen’, vorgesehen ist. Im obigen Beispiel kann ein Werkzeug zur Aufbereitung der in SGML vorliegenden Daten folglich nicht zwischen „Jerry Moonbeam“ und „Moonbeam Jerry“ unterscheiden, obwohl

hier eindeutig eine Unterscheidung in Vor- und Nachname (des Beinamens) getroffen werden kann.

Obwohl dieses Tag-Set ein Grundgerüst zur Annotation von Namen bereitstellt, ist es für eine standardisierte Auszeichnung aller Bestandteile von komplexen Namensausdrücken nicht unbedingt geeignet, da Attributwerte wie die bereits erwähnten `epithet`, `office`, `military` oder auch `nobility` zwar als Beispielwerte in Erscheinung treten, jedoch nicht in der entsprechenden DTD definiert sind. In der Realität bedeutet dies, daß jeder Benutzer dieses Tag-Sets zwar die grundlegenden SGML-Elemente (vgl. Tabelle 3) einsetzt, jedoch eigene Annotationsschemata für die Werte verschiedener Attribute entwickelt. Das Erarbeiten eines Tag-Sets zur möglichst spezifischen Auszeichnung beliebiger Elemente von Namensausdrücken hätte gewiss den Rahmen der TEI gesprengt, jedoch wäre eine weniger generelle Behandlung von Namen als die hier propagierte vor allem für Standardisierungszwecke wünschenswert gewesen.

Der Einsatz dieses standardisierten Tag-Sets im `pronto`-System zur Bereitstellung unserer Daten in einem möglichst generellen Format ist prinzipiell möglich. Hierzu müßte lediglich mit einer Skriptsprache wie `awk` oder `sed` (vgl. beispielsweise Dougherty und Robbins, 1997; Herold, 1994) ein Programm entwickelt werden, das das von uns benutzte Annotationsschema in die oben beschriebenen SGML-Elemente konvertiert. Die bereits erwähnten Probleme des TEI Tag-Sets treten dabei unmittelbar zu Tage: Wie sollen die von uns eingesetzten Berufsbezeichnungen markiert werden? Denkbar wäre an dieser Stelle der Einsatz der Elemente `<roleName>` oder `<addName>`.[9] Keinesfalls werden wir auf gewisse vorgeschlagene, jedoch nicht definierte Elemente bzw. Attribute dieses Tag-Sets zurückgreifen können, die nur durch Weltwissen instanziiert werden können. Hierzu zählen die Kennzeichnungen von religiösen Namen wie „Muhammad Ali“, Mädchennamen wie „Roberts“ (im Gegensatz zu „Thatcher“) oder die Unterscheidung von realem Namen und Spitznamen.

4.3.2. Automatische Auswertung

4.4. Die Wortlänge als Hinweis auf einen Namen?

Neben lexikalischen Informationen aus den Namenslisten (Abschnitt 4.2) und kontextuellen Informationen (Abschnitt 3.1) wollten wir ursprünglich auch die Länge einer Kette als Hinweis auf einen Namen zur Hilfe nehmen. Die Prämisse für eine solche Herangehensweise ist natürlich ein signifikanter Unterschied zwischen der Länge von Namen und der Länge von ‘herkömmlichen’ Nomina. Als Namenslisten haben wir für unsere Versuche die beiden vollständig abgeglichenen Listen mit 2.225 Vor- und 13.566 Nachnamen sowie ein von allen Namen befreites[10] Korpus verglichen.

Abbildung 1 on the following page zeigt die Ergebnisse unserer Versuche: Vor- und Nachnamensliste verhalten sich sehr ähnlich, was die statistische Verteilung der Länge der jeweiligen Einträge angeht, so sind je 21,4% aller Vornamen fünf bzw. sechs Zeichen

[9] Die Autoren von Kapitel 20 der TEI Richtlinien merken zu Recht an: „The distinction [between `<roleName>` and `<addName>`] is not always a clear one.“ (Sperberg-McQueen und Burnard, 1994, S. 588)

[10] Soweit unser System zum Untersuchungszeitpunkt dazu in der Lage war.

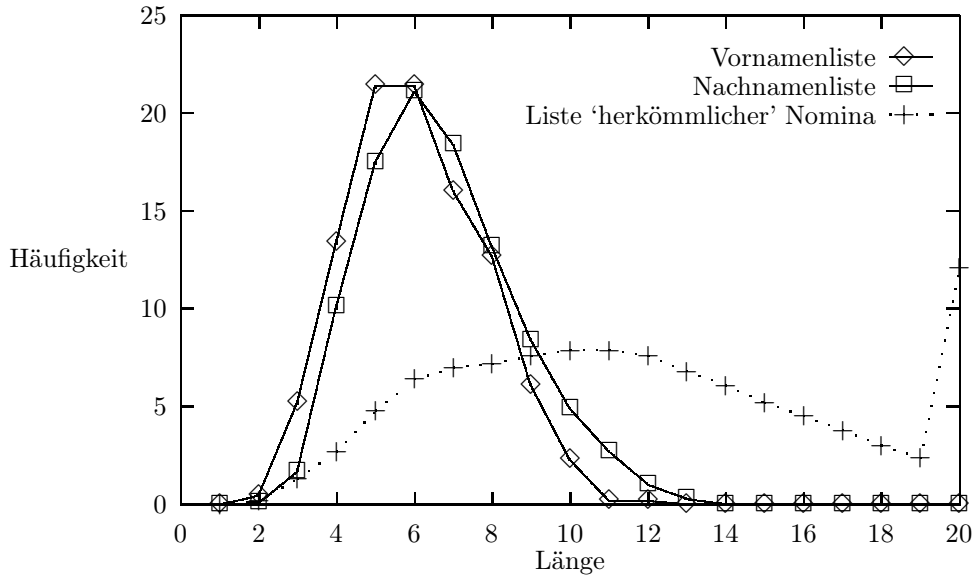

Abbildung 1: Relation zwischen Wortlänge und Häufigkeit im Korpus

lang, wohingegen 21,1% aller Nachnamen sechs Zeichen lang sind. Vor und hinter diesen Punkten nimmt die Kurve rapide ab, Namen mit mehr als 14 Zeichen Länge tauchen in unseren Listen überhaupt nicht auf. Auf der Grundlage dieser Daten kann man beispielsweise eine Menge von Regeln in den Tagger integrieren, die bei einem unbekannten Wort mit einer Länge von mehr als 14 Zeichen, das in einem unbekannten Kontext auftaucht, die Hypothese aufstellen, daß es sich bei diesem Wort höchstwahrscheinlich nicht um einen Namen handeln kann.

Die statistische Verteilung der 'herkömmlichen' Nomina ist beinahe linear, wobei Worte mit einer Länge zwischen 14 und 19 Zeichen seltener vorkommen als solche mit sechs bis 12 Zeichen Länge. Der Anstieg der Kurve bei einer Länge von 20 Zeichen erklärt sich dadurch, daß dieser Meßpunkt alle Worte umfaßt, die 20 Zeichen oder mehr lang sind.

5. Verwandte Arbeiten

5.1. Kontextbasierte Erkennung: Mani & MacMillan

Der Namenstagger von Mani und MacMillan (1996) ist architekturell nicht unähnlich zu `pronto`, hat jedoch, wie in der Einleitung erwähnt, eine ausgeweitete Funktionalität und arbeitet, im Gegensatz zu `pronto`, ohne Namenslisten, sondern verläßt sich ausschließlich auf Kontextinformationen. Die zusätzliche Funktionalität besteht darin, daß neben dem bloßen Erkennen von Namen auch noch Koreferenzen zwischen verschiedenen Erwähnungen (*mentions*) desselben Objekts erkannt werden. Damit lassen sich einfache Repräsentationen von Diskursstrukturen aufbauen.

Wie das System von Göser (1997) kann dieser Tagger nicht nur Personennamen, sondern auch weitere Arten von Eigennamen, also z. B. Firmen- oder Ortsnamen erkennen. Anders als `pronto` ist dieses System für englischsprachige Texte ausgelegt, was die Erkennung von Namen allein wegen der selteneren Großschreibung im Englischen erleichtert.

Mani und MacMillan verfahren so, daß im ersten Durchlauf des Systems mögliche Kandidaten für Namen gefunden werden. Diese Kandidaten sind möglichst kurze Ketten, die – bei späteren Durchläufen – zu längeren Ketten zusammengefügt werden. Ob eine Kette ein Name ist, wird von verschiedenen Modulen festgestellt, den *knowledge sources* (KS). Diese Module sind architekturell mit den Komponenten in `pronto` vergleichbar. Die Autoren führen allerdings nicht aus, ob die KS serialisiert auf den Text zugreifen oder ob alle KS bei einem einzigen Durchlauf aktiv werden. Beispiele für *knowledge sources* sind z. B. Kontextchecker für Firmennamen, Aktionsverben und Honorifica. Diese Module verfügen über Wortlisten, die typische Worte enthalten, wie sie im Kontext eines Kandidaten auftreten können. Das Modul für die Firmennamen kennt u. a. Begriffe wie *Inc.*, *Bank of...* oder das Kaufmannsund „&“. Die Aktionsverben, die in englischen Texten ebenfalls in der Nähe von Eigennamen zu finden sind[11], bilden weitere Indikatoren für Eigennamen. Da Mani und MacMillan nur Zeitungstexte behandeln, sind diese Verben als Indikatoren gut geeignet. Als Beispiele für Honorifica werden *Mr.* oder *His Holiness* aufgeführt. Die KS berechnen weiterhin Wahrscheinlichkeiten über die Gültigkeit der so gemachten Hypothesen, auch hier ist eine Parallelität zu `pronto` zu erkennen. Eine weitere KS identifiziert Appositionen. Strukturen der Form <NP,NP> sind als Kandidaten für Eigennamen interessant, da sie häufig verwendet werden, um neue Eigennamen einzuführen. In einer solchen Konstruktion kann ein Eigenname entweder auf der linken Seite des Kommas auftreten (<name>, <organization>*'s top manging director*), oder dem Komma auf der rechten Seite folgen (*a top Japanese executive,* <name>). Zum Finden der Appositive setzen Mani und MacMillan reguläre Ausdrücke ein. Diese haben sich für diesen Zweck als zuverlässiger erwiesen als z. B. das Parsing eines Textes.

Ist eine Kette von den KS als Eigenname erkannt worden, erhält sie den Status eines *mention*. Für jeden *mention* wird eine Art Diskursreferent, ein *peg*, eingeführt. Dazu werden die *mentions* in eine normalisierte Form überführt. Diese besteht z. B. aus dem Nachnamen bei Personen, oder dem Firmennamen ohne den Bezeichner der Art der Gesellschaft (der kanonische Name von *Bill Clinton* und *President Clinton* ist *Clinton*, der von *IBM Inc.* ist *IBM*). Alle bisher gefundenen *pegs* mit demselben normalisierten Namen sind mögliche Antezedenten eines neu eingeführten *pegs*. Da dadurch nicht alle bisher gefundenen *pegs* in Frage kommen, ist es leichter, die irrelevanten Antezedenten auszublenden, um den Suchraum für Koreferenzen möglichst klein zu halten. Wenn bislang keine identischen normalisierten Namen gefunden wurden, nutzt das System eine *nearest neighbor* Strategie, um einen Antezedens zu finden. Dabei werden alle *mentions* in Betracht gezogen, die in unmittelbarer Nähe zu finden sind. Diese werden aufgrund der Ähnlichkeit zum aktuellen *peg* bewertet. Der Nachbar mit den meisten identischen

[11] Da die Möglichkeiten der indirekten Sprache im Englischen aufgrund des fehlenden Konjunktivs nicht so vielfältig sind wie im Deutschen, werden – gerade in Zeitungstexten – sehr viele Wörter verwendet, die Sprechakte beschreiben (Bergler, 1995).

Worten im Namen wird als bester Kandidat für Koreferenz bewertet und als Antezedens festgelegt. Im anschließenden *matching* Prozeß werden Hypothesen über die Koreferenz der gefundenen *pegs* aufgestellt. Verschiedene Algorithmen untersuchen die *pegs* auf Koreferenz und sammeln dabei Informationen über diese auf, z. B. Geschlecht und Position in einer Organisationsstruktur. Durch dieses zusätzliche Wissen können *pegs*, die nicht denselben normalisierten Namen tragen, aufeinander bezogen werden. Außerdem können solche *pegs*, die zwar denselben normalisierten Namen tragen, sich aber nicht auf dasselbe Objekt beziehen, voneinander abgegrenzt werden. *Hillary Clinton* und *Bill Clinton* tragen den identischen normalisierten Namen *Clinton*, sind aber verschiedene Objekte. Durch Zusatzinformation wie *gender* oder *president* oder *first lady* im Feld *occupation* kann eine fälschlich angenommene Koreferenz zurückgenommen werden (vgl. Abb. 3.4 in Mani und MacMillan, 1996, S. 56).

Mani und MacMillan zeigen ein innovatives System zur Identifikation von Eigennamen in Zeitungstexten. Die erreichten Ergebnisse (73 % Recall und 88 % Precision) sind zwar nicht so gut wie die Ergebnisse anderer Systeme (siehe Abschnitt 1), doch man muß den Autoren zugute halten, daß sie eine wesentlich erweiterte Funktionalität zur Verfügung stellen. Sie erwähnen nicht, ob sich die genannten Zahlen nur auf die reine Erkennungsleistung beziehen, oder ob der gesamte Prozeß, inklusive Koreferenzierung, diese Ergebnisse liefert. Besonders die Erfolge bei der Bewertung der Appositionen zeigen, daß die Einbeziehung von Kontext in Systemen zum Taggen von Eigennamen eine lohnende Erweiterung ist. Bei der Weiterentwicklung von `pronto` ist Kontext der Indikator, der am ehesten ausgenutzt werden sollte. Es fällt allerdings auf, daß Mani und MacMillan zwar die Verwendung von Namenslisten ablehnen, daß jedoch viele der verwendeten *knowledge sources* ihr Wissen aus Listen ziehen – seien es nun Listen von Honorifica oder Gesellschaftsbezeichnern. Die hier verwendeten Listen sind natürlich kleiner und somit wartbarer als große Namenslisten, wie sie in `pronto` verwendet werden, doch die in `pronto` implementierte Datenstruktur eines tries zeigt, daß auch der Zugriff auf sehr große Listen effizient möglich ist.

6. Ausblick

Erste noch sehr vorläufige Evaluierungsergebnisse zeigen, daß ein erhebliches Optimierungspotential vorhanden ist. Insbesondere die Verwertung von Negativ-Informationen („Stoppwortlisten“), die automatisch aus dem Vergleich der `pronto`-Resultate mit manuell annotierten Texten extrahiert werden können, dürfte es erlauben, die Erkennungsresultate mit relativ geringem Aufwand noch deutlich zu verbessern. Weiterhin ist die Ermittlung der Wahrscheinlichkeit, die die einzelnen Komponenten angeben, noch stark verbesserungswürdig. Die zur Zeit vergebenen Wahrscheinlichkeiten basieren auf der Intuition der Autoren. Die Algorithmen sind jedoch offen für komplexere Berechnungen, sofern ausreichendes Datenmaterial dafür vorliegt.

Die ersten Testläufe zeigen aber auch, daß sich bei Zeitungstexten gegenwärtig Werte von jeweils 70 % Recall und Precision erzielen lassen. Durch seine modulare Struktur ist es möglich, das `pronto`-System so zu konfigurieren, daß die Erkennungsergebnisse entweder in Richtung auf höhere Präzision (auf Kosten des Recalls) oder umgekehrt

optimieren lassen. Durch eine entsprechende Konfiguration konnten bereits Werte über 90 % für beide Maße erzielt werden.

A. Bestandteile komplexer Namensausdrücke

Die folgenden Listen enthalten Adelsbezeichnungen (Titel und Präpositionen), militärische Titel bzw. Rangbezeichnungen und akademische Titel, die in `pronto` zur Disambiguierung eingesetzt werden (vgl. Abschnitt 3.1).

- Adelsbezeichnungen
  - Vor dem Vornamen
    Kaiser, Kaiserin, König, Königin, Prinz, Prinzessin, Herzog, Herzogin, Fürst, Fürstin, Graf, Gräfin, Freiherr, Freifrau, Baron, Baronin, Ritter, Edler, Edle
  - Nach dem Vornamen
    von, Von, von der, von den, Von den, Von der, von und zu, vom und zum, van, Van, van den, van de, Van den, Van de, de, De
- Militärische Titel und Rangbezeichnungen
  - Deutschland
    - Heer und Luftwaffe
      General, Generalleutnant, Generalmajor, Brigadegeneral, Major, Oberstarzt, Oberstleutnant, Generalarzt, Oberfeldarzt, Oberst, Oberstabsarzt, Hauptmann, Stabsarzt, Oberleutnant, Leutnant, Oberstabsfeldwebel, Gefreiter, Stabsfeldwebel, Hauptfeldwebel, Oberfähnrich, Stabsunteroffizier, Oberfeldwebel, Unteroffizier, Feldwebel, Fähnrich, Fahnenjunker, Hauptgefreiter, Obergefreiter, Soldat
    - Marine
      Kapitän zur See, Admiral, Vizeadmiral, Konteradmiral, Flottillenadmiral, Admiralarzt, Flottenarzt, Fregattenkapitän, Flottillenarzt, Korvettenkapitän, Marineoberstabsarzt, Kapitänleutnant, Marinestabsarzt, Oberleutnant zur See, Leutnant zur See, Oberstabsbootsmann, Stabsbootsmann, Hauptbootsmann, Oberfähnrich zur See, Oberbootsmann, Bootsmann, Fähnrich zur See, Obermaat, Maat, Seekadett, Hauptgefreiter, Obergefreiter, Gefreiter, Matrose
  - Schweiz
    General, Oberstkorpskommandant, Oberdivisionär, Oberstbrigadier, Oberst, Oberstleutnant, Major, Hauptmann, Oberleutnant, Leutnant, Adjutant, Feldweibel, Fourier, Wachtmeister, Korporal, Gefreiter, Soldat
  - Österreich
    General, Generalmajor, Oberst, Oberstleutnant, Major, Hauptmann, Rittmeister, Oberleutnant, Leutnant, Stabswachtmeister, Stabsfeuerwerker, Wachtmeister, Feuerwerker, Zugführer, Korporal, Gefreiter, Vormeister, Infanterist

- Akademische Titel
  - Generelle Zusätze
    mult., eh., e. h., E. h., h. c., h. c. mult., habil., Dres.
  - Die Titel im einzelnen
    Dr. pharm., Dr. rer. mont., Dr. mont., Dr. nat. techn., Dr. forest., Dr. rer. hort., Dr. rer. comm., Dr.-Ing., Dr. sc. agr., Dr. agr., Dr. sc. math., Dr. med., Dr. med. univ., Dr. phil. nat., Dr. rer. nat., Dr. sc. nat., Dr. päd., Dr. phil., Dr. jur., Dr. iur., Dr. j. u., Dr. jur. utr., Dr. disc. pol., Dr. rer. soc. oec., Dr. rer. camer., Dr. rer. pol., Dr. sc. pol., Dr. oec. publ., Dr. rer. techn., Dr. scient. techn., Dr. sc. techn., Dr. techn., Dr. techol., D., D. theol., Dr. med. vet., Dr. oec., Dr. rer. oec., Dr. med. dent.

Literatur

Bergler, Sabine (1995): "Generative Lexicon Principles for Machine Translation: A Case for Meta-Lexical Structure". *Machine Translation* 9: S. 155–182.

Church, Kenneth W. und Mercer, Robert L. (1993): "Introduction to the Special Issue on Computational Linguistics Using Large Corpora". *Computational Linguistics* 19 (1): S. 1–24.

Coling (1994): *COLING 94 – The 15th International Conference on Computational Linguistics*, Kyoto, Japan. Association for Computational Linguistics. 2 Bände.

Crystal, David (1995): *Die Cambridge-Enzyklopädie der Sprache*. Frankfurt/Main, New York: Campus. Sonderausgabe 1998. Köln: Parkland Verlag.

D-Info 2 (1995): "D-Info 2". CD-ROM. TopWare, Mannheim.

Dougherty, Dale und Robbins, Arnold (1997): *sed & awk*. A Nutshell Handbook. Cambridge, Köln, Paris etc.: O'Reilly & Associates, 2. Auflage.

Gancarz, Mike (1995): *The UNIX Philosophy*. Digital Press.

Goldfarb, Charles F. (1990): *The SGML Handbook*. Oxford: Oxford University Press.

Göser, Sebastian (1997): "Inhaltsbasiertes Information Retrieval: Die TextMining-Technologie". *LDV-Forum* 14/1.

Gotthard, Willi und Seiffert, Roland (1997): "Text Mining White Paper". IBM Deutschland Entwicklung GmbH.

Herold, Helmut (1994): *awk und sed*. Bonn, Reading, Menlo Park, etc.: Addison-Wesley, 2. Auflage.

Herold, Helmut (1995): *lex und yacc*. Bonn, Reading, Menlo Park, etc.: Addison-Wesley, 2. Auflage.

Ide, Nancy (1994): "Encoding Standards for Large Text Resources: The Text Encoding Initiative". In: Coling (1994), S. 574–578. 2 Bände.

Kernighan, Brian W. und Pike, Rob (1986): *Der UNIX-Werkzeugkasten*. München, Wien: Hanser.

Leech, Geoffrey; Garside, Roger und Bryant, Michael (1994): "CLAWS4: The Tagging of the British National Corpus". In: Coling (1994), S. 622–628. 2 Bände.

Levine, John R.; Mason, Tony und Brown, Doug (1992): *lex & yacc*. A Nutshell Handbook. Cambridge, Köln, Paris etc.: O'Reilly & Associates, 2. Auflage.

Mani, Inderjeet und MacMillan, T. Richard (1996): "Identifying Unknown Proper Names in Newswire Text". In: *Corpus processing for lexical acquisition*, herausgegeben von Pustejovsky, James und Boguraev, Branimir, Cambridge: M.I.T. Press.

McKelvie, David; Brew, Chris und Thompson, Henry (1997): "Using SGML as a Basis for Data-Intensive NLP". In: *Proceedings of ANLP 97*. Association for Computational Linguistics, Washington D. C. Online verfügbar: `http://www.ltg.hcrc.ed.ac.uk/~dmck/anlp97.ps`.

PND (1996): *Normdaten-CD-ROM für Personennamen und Schlagwörter*. Deutsche Bibliothek, Frankfurt am Main. Enthält Handbuch.

Ronthaler, Marc und Sauer, Uli (1997): "Osiris – Computerlinguistik in der wissenschaftlichen Bibliothek". Vortrag im Rahmen der DGfS/CL '97. Online verfügbar:`http://www.cl-ki.uni-osnabrueck.de/~marc/publications.h%tml`.

Sperberg-McQueen, C. M. und Burnard, Lou (Herausgeber) (1994): *Guidelines for Electronic Text Encoding and Interchange*. Chicago, Oxford: University of Chicago, University of Oxford. Version P3, 2 Bände.

Steiner, Petra (1995): "Anforderungen und Probleme beim Taggen deutscher Zeitungstexte". In: *Lexikon und Text*, herausgegeben von Hinrichs, Helumd und Hinrichs, Erhard, Tübingen: Niemeyer.

Volk, Martin (1998): "Markup of a Test Suite with SGML". In: *Linguistic Databases*, herausgegeben von Nerbonne, John, Cambridge: Cambridge University Press, Nummer 77 in CSLI Lecture Notes, S. 59–76. Online verfügbar: `http://www.ifi.unizh.ch/CL/volk/papers/SGMLMarkup.ps.gz`.

Wauschkuhn, Oliver (1995): "The Influence of Tagging on the Results of Partial Parsing in German Corpora". In: *Proceedings of the 4th Int. Workshop on Parsing Technologies*. Prag.