Ludwig-Maximilians-Universität München
Centrum für Informations- und Sprachverarbeitung

SS 2004

Magisterarbeit im Studiengang Computerlinguistik

Automatische Adreßerkennung – ein Ansatz für deutsche Adressen und seine Implementierung

Wolfgang A. Mederle
Prinzregentenstraße 124/IV
D-81677 München

<reboot@cis.uni-muenchen.de>
<URL:http://www.mederle.de/>

27. September 2004

Betreut durch Prof. Franz Guenthner

Diese Magisterarbeit wurde auf einem Apple PowerBook G4 unter OS X 10.3 mit Hilfe von KOMA-Script, PDFLaTeX, GNU Emacs und TeXshop gesetzt. Referenzen, Zitatverweise und URLs sind anklickbar, sofern dies nicht die Papierform der Arbeit ist.

Inhaltsverzeichnis

Vorwort

Der Mensch, dessen Wahrnehmung auf Mustererkennung optimiert ist, erkennt schon mit einem flüchtigen Blick, ob es sich bei einem Stück Text um eine Adresse handelt. Er braucht dazu kaum Informationen, muß von dem angegebenen Ort oder gar dem Adressaten noch nie gehört oder gelesen haben. Jeder Leser dieser Arbeit, der die Titelseite nur kurz überflogen hat, hat einen Teil davon als Adresse erkannt – auch wenn er sich, an dieser Stelle angekommen, nicht mehr daran erinnern kann, *welche* Adresse es war. Dazu war es nicht nötig, die Adresse als solche zu kennzeichnen.

Ein Computer vergißt eine Adresse nie mehr. Er ist auch in der Lage, sich Tausende von Adressen in Sekundenbruchteilen zu merken und nie mehr zu vergessen (ein funktionierendes Backup vorausgesetzt). Ungleich schwerer ist es für ihn aber, selbständig Adressen zu erkennen. Er kann nicht einmal *zwei*dimensional sehen, ein Text ist für ihn nur eine Kette von Nullen und Einsen wie alles andere auch.

Bislang werden Adreßverzeichnisse für Branchenbücher oder Firmendatenbanken in der Regel noch manuell erstellt. Dabei schleichen sich unweigerlich Fehler ein: ständiges Wiederholen einer langweiligen Tätigkeit führt zu nachlassender Konzentration; dazu addiert sich Unwissenheit etwa über die korrekte Rechtschreibung. Für solche Aufgaben eignet sich ein Rechner, der ohne zu Murren dieselbe Funktion millionenfach ausführt, wesentlich besser. Der Mensch kann sich dann darauf beschränken, die hoffentlich wenigen Zweifelsfälle zu bearbeiten, die umso seltener auftreten, je besser dem Computer das Konzept „Adresse“ beschrieben wurde.

Automatische Adreßerkennung kann auch dazu dienen, Informationen auf neuartige Weise zu verknüpfen. Suchmaschinen, die Adreßinformationen auswerten, ermöglichen eine ortsbezogene Suche („Zeig mir alle koreanischen Restaurants im Umkreis von 20 Kilometern!“). Bei der Produktrecherche im WWW kann das Vorhandensein einer Adresse auf einem Website – das für deutsche Firmen vorgeschrieben ist – einen Hinweis auf die Seriosität des Anbieters liefern oder helfen, den Site des Herstellers ausfindig zu machen, der seinen Namen in der Regel in der Firmenadresse nennen wird.

Während dem Menschen die äußere Form (meist schon die Silhouette) genügt, um eine Adresse als solche zu identifizieren, muß man den Rechner mit exakten Informationen versorgen, was eine Adresse ausmacht. Einen Ansatz dazu stellt diese Arbeit vor.

1 Eine Grammatik deutscher Adressen

1.1 Normen und Konventionen

Kommunikation basiert in vielerlei Hinsicht auf Konventionen, die sich im Laufe der Zeit entwickelt haben und weiterentwickeln. In stark arbeitsteiligen Gesellschaften ist es nötig, Konventionen in bestimmten Bereichen weiter zu spezifizieren und in Normen zu gießen.

Sprache an sich läßt sich schlecht normieren; Rechtschreibung und auch Grammatik sind Konventionen, die beschrieben, nicht bestimmt werden, und unterliegen einem beständigen Wandlungsprozeß. Normen und De-facto-Normen wie die Duden-Rechtschreibung dienen der Vereinheitlichung und Vermeidung von Ambiguitäten.

Hinweise zur Notation

Grammatiken werden im Folgenden in einer Art DCG[a]-Notation angegeben. Dabei wird zwischen Terminal- und Nonterminalsymbolen unterschieden. Nonterminale werden in `Festbreitenschrift` angegeben, Terminale in *Kursivschrift*. Eine Regel der Form `A` → `B` c bedeutet, daß das Nonterminal `A` in das Nonterminal `B` und das Terminal c expandiert.
Reguläre Ausdrücke entsprechen der Perl-Syntax und werden ebenfalls in Festbreitenschrift dargestellt.

[a] Definite Clause Grammar

Die einfachste heute übliche Adresse hat die Form

```
Adresse → Name
          Straße
          Ort
Straße → Straßenname Hausnummer

Ort → Postleitzahl Ortsname
```

Diese Form existiert weniger lang, als man meinen könnte. Hausnummern waren bis ins 19. Jahrhundert nicht überall üblich, sondern Häuser waren meist durch Symbole gekennzeichnet oder trugen Namen. Der erste Zahlencode für Ortsnamen wurde 1853 von der Postverwaltung der Thurn und Taxis verwendet. Die heute gültigen fünfstelligen Postleitzahlen wurden im Juli 1993 im Zuge der Ost-West-Vereinheitlichung des Postleitzahlsystems eingeführt; sie lösten ein vierstelliges System ab, das wiederum der Nachfolger eines zweistelligen Zahlensystems gewesen war.

1.1.1 DIN 5008

Adressen sind ein elementarer Bestandteil der Geschäftskommunikation. Sie müssen eindeutig sein und von jedem, der mit ihnen in Kontakt kommt, verstanden werden. Wie sich ein Mechaniker darauf verlassen können muß, daß eine M8-Schraube – egal, von welchem Hersteller – in ein M8-Gewinde paßt, muß auch der Verfasser eines Briefes oder der Autor eines Website sicher sein können, daß die Adresse, die er verwendet oder angibt, an keiner Stelle des Zustellweges falsch ausgelegt wird.

Zu diesem Zweck gibt es die DIN[1] 5008: „Schreib- und Gestaltungsregeln für die Textverarbeitung“ – früher: „Regeln für Maschinenschreiben“. Die letzten Änderungen an dieser Norm betreffen insbesondere die Formatierung von Adreßangaben, und die Entwicklung dieses Teils der Norm ist nicht nur aus sprachgeschichtlicher Perspektive interessant, sondern kann auch für die automatische Erkennung von Bedeutung sein, wenn es etwa um die Erfassung historischer Dokumente über OCR[2] und anschließende Weiterverarbeitung geht.

Die DIN 5008, Stand 1963, macht folgende Angaben zur Adreßformatierung für „Geschäftliche, behördliche und andere dienstliche Schreiben“[ADO1974, 171ff.]:

> **Zeilenabstand.** Es wird mit einfachem Zeilenabstand geschrieben. In Schriftstücken mit hoch- oder tiefgestellten Schriftzeichen und in Schreiben besonderer Art (z. B. Glückwunschschreiben, Zeugnissen) darf man mit größerem Zeilenabstand schreiben. [. . .]
>
> **Anschrift des Empfängers.** Anschriften werden im Anschriftfeld aller Schriftstücke und auf Briefhüllen in gleicher Weise geschrieben. Satzzeichen innerhalb einer Anschriftzeile werden geschrieben; am Zeilenende fallen sie weg. [. . .]
>
> *Leerzeilen.* Zwischen den aufeinanderfolgenden Bestandteilen der Anschrift bleibt je eine Leerzeile, also zwischen
> Beförderungs-, Aushändigungsvermerk;
> Empfängerbezeichnung;
> Postleitzahl, Bestimmungsort (ggf. Nummer des Zustellpostamtes), Postfach- oder Wohnungsangabe (In Briefanschriften an Postfachinhaber soll unter dem Bestimmungsort nur das Postfach angegeben werden.);
> Bestimmungsland (bei Schreiben an Empfänger im Ausland).
>
> *Unterstrichen* werden Beförderungs- und Aushändigungsvermerke, Bestimmungsort (mit Postleitzahl und postalischen Zusätzen) und Bestimmungsland.
>
> *Empfängerbezeichnungen* sollen sinngemäß in Zeilen aufgeteilt werden.
>
> *Berufs- oder Amtsbezeichnungen* werden in der Regel neben „Herrn“, „Frau“, „Fräulein“ (nicht abkürzen), längere unter dem Namen geschrieben.

[1] Deutsche Industrie-Norm, Deutsches Institut für Normung, Das Ist Norm; die Bedeutung der Abkürzung DIN unterliegt ebenso dem Wandel wie die Inhalte der Normen

[2] Optical Character Recognition, Texterkennung in Bilddaten, zum Beispiel Scans

Akademische Grade (wie Dr., Dipl.-Hdl., Dipl.-Ing., Lic. theol.) stehen im allgemeinen vor dem Namen als dessen Bestandteil.

Bei *Firmenanschriften* wird das Wort „Firma“ weggelassen, wenn aus der Empfängerbezeichnung erkennbar ist, daß es sich um eine Firma handelt.

Bestimmungsort. Die Postleitzahl beginnt an der Fluchtlinie. Im allgemeinen werden nur kurze Ortsnamen gesperrt.

Auslandsanschriften können der im Empfängerland üblichen Form angepaßt werden. [. . .]

Musteranschriften
Die Nummern 1 bis 9 vor dem Zeilenanfang zeigen die Stellung der Anschrift im Anschriftfeld, das nach den Vordrucknormen Platz für eine neunzeilige Anschrift hat.

(1)
```
1
2
3 Fräulein
4 Erika Werner
5
6 858 Bayreuth
7 Bahnhofstr. 4/6
8 bei Müller
9
```

(5)
```
1 Durch Eilboten - auch nachts
2
3 Frau Studienrätin
4 Dipl.-Hdl. Dr. Eva Meier
5
6 8803 Rothenburg
7 Villa Talblick
8
9
```

(6)
```
1 Einschreiben - mit Rückschein
2
3 Herrn Rechtsanwalt
4 Dr. Otto Freiherr von Bergheim
5
6 8 München 27
7 Leonhard-Eck-Str. 7 III
8
9
```

(10)
```
1
2
3 Wäschegroßhandel
4 Robert Bergmann
5
6 5 K ö l n 17
7 Bonner Straße 80 - 82
8
9
```

(11)
```
1
2
3 Firma
4 Otto Pfleiderer
5
6 7969 Braunenweiler
7
8
9
```

(12)
```
1
2
3 Amtsgericht Leer
4 Grundbuchamt
5
6 295 L e e r
7
8
9
```

(13)
```
1
2
3 Nassauische Heimstätte GmbH
4 Abt. Landestreuhandstelle
5
6 6 Frankfurt 10
7 Postfach 4711
8
9
```

Neben Hilfsformatierungen wie Unterstreichung und gesperrter Druck, die auf die Beschränkungen mechanischer Schreibmaschinen zurückzuführen sind, ist der augenfälligste Unterschied zu heutigen Adreßangaben, daß die Straße *nach* dem Ort angegeben wird. Außerdem wird vor der unterstrichenen Ortsangabe eine Leerzeile eingefügt – der Grund

dafür dürfte die manuelle Briefsortierung gewesen sein; sowohl die Leerzeile als auch die Unterstreichung unterstützen die menschliche Mustererkennung und erleichtern das Auffinden der Orts- und Straßenangabe.

In „Bürokommunikation – Schreiben und Gestalten nach DIN 5008“ von 1996 sind einige Änderungen zur Norm von 1963 feststellbar [Hovermann, Registerpunkt „Anschriftfeld“]:

> Das Anschriftenfeld hat die Maße 40 mal 85 Millimeter. Für die Anschrift stehen Ihnen neun Zeilen zur Verfügung.
>
> Beispiel:

Zeile 1: Einschreiben
Zeile 2:
Zeile 3: Herrn
Zeile 4: Professor Dr. Joachim Müller
Zeile 5: Am Deich 13
Zeile 6:
Zeile 7: 12121 Emden
Zeile 8:
Zeile 9:

> Beispiel:

Zeile 1: Nicht nachsenden
Zeile 2: Einschreiben - Rückschein
Zeile 3: Herrn
Zeile 4: Professor Dr. Joachim Müller
Zeile 5: Universität Emden
Zeile 6: Abt. 2/34 II
Zeile 7: Am Deich 13
Zeile 8:
Zeile 9: 12121 Emden

> Vorausverfügungen und besondere Sendungsarten werden unterstrichen, wenn sie aufgrund Platzmangels nicht mit einer Leerzeile von der Anschrift abgesetzt werden können.
>
> Beachten Sie:
> Postleitzahlen werden nicht gesperrt geschrieben. Ortsnamen werden nicht fett geschrieben. Akademische Grade, wie zum Beispiel Dr. und Professor stehen unmittelbar vor dem Namen, also in der gleichen Zeile.
>
> Beispiel:

Herrn Dr. Arnold Hülsley

> Berufsbezeichungen wie zum Beispiel Direktor, Rechtsanwalt usw. werden hinter ”Herrn” beziehungsweise ”Frau”geschrieben.

[...]
Bei Untermietern wird der Name des Wohnungsinhabers unterhalb des Namens des Empfängers, also des Untermieters geschrieben werden (sic!).
Beispiel:

Frau
Silke Klein
bei Max Müller

Bei Firmenbezeichnungen entfällt die Bezeichnung Firma, wenn aus der Firmenbezeichnung eindeutig zu erkennen ist, daß es sich um eine Firma handelt.

Der Duden von 1986, der ansonsten die gleichen Angaben zur Adreßformatierung macht, präzisiert weiter: „Der Bestimmungsort [wird] nicht unterstrichen.“ [Duden, 67]. Zusätze wie „bei Müller“ folgen direkt nach dem Namen, die Straße steht nun *vor* dem Ort, der immer noch durch eine Leerzeile vom restlichen Adreßblock abgetrennt werden soll.

Die letzte Revision der DIN 5008 stammt vom November 2001. Erst vor kurzem gab es eine außerplanmäßige Änderung, die in [BVA2004] mit „akutem Änderungsbedarf“ begründet wird und den Namen „DIN 5008/A1“ trägt. Dieser Änderungsbedarf ergibt sich aus der gewünschten Angleichung an internationale Schreibweisen und Schaffung von „Platz für größere Zusätze und Vermerke einschließlich elektronischer Frankierung.“ [BVA2004, 1] Die Neudefinition des Anschriftfeldes unterteilt dies in eine Zusatz- und Vermerkzone von drei Zeilen, die genügend Platz für „elektronische Freimachungsvermerke (z. B. über ≪Stampit≫)“ [BVA2004, 2] zur Verfügung stellt, und eine Anschriftzone in den verbleibenden sechs Zeilen.

Erstmals werden Änderungen in der Schriftgröße und -art vorgegeben, falls die Zusatzzone zu klein für elektronische Frankierverfahren ist: „Werden in der Anschriftzone alle sechs Zeilen benötigt, ist dort die Schriftgröße zu reduzieren. Eine Schriftgröße von 8 Punkt darf aber nicht unterschritten werden. Bei Schriftgrößen kleiner als 10 Punkt sind zudem serifenlose Schriften wie Arial oder Helvetica zu verwenden (also keine Serifenschriften wie Times).“ [BVA2004, 3]

Darüberhinaus werden folgende Formatierungsanweisungen gegeben [BVA2004]:

Der Verzicht auf Leerzeilen Im gesamten Anschriftfeld werden keine Leerzeilen geschrieben, also weder zur Trennung der Zusätze und Vermerke von der Anschrift, noch innerhalb der Anschrift vor dem Ortsnamen. Daraus folgt auch, dass der Text in der Zusatz- und Vermerkzone nur dann in der ersten Zeile beginnt, wenn er drei Zeilen umfasst. Ein zweizeiliger Text beginnt in der zweiten Zeile, ein einzeiliger in der dritten. [...]

Die Hervorhebungen Es gibt keine Hervorhebungen im Anschriftfeld mehr. Bisher sollten postalische Vermerke bei fehlender Leerzeile unterstrichen werden.

Die Satzzeichen Präzisiert wurde die Regel für Satzzeichen: Innerhalb der Anschrift werden Satzzeichen innerhalb einer Zeile, jedoch nicht am Zeilenende geschrieben. Bei Zusätzen und Vermerken dürfen jedoch zusätzlich auch Satzzeichen am Zeilenende stehen, weil dies für bestimmte Vorausverfügungen erforderlich ist, die alle mit Ausrufezeichen schließen (z. B. „Nicht nachsenden!“).

Das „Professorenproblem“ Die Grundregel wurde nicht verändert. Danach werden Berufs- und Amtsbezeichnungen neben „Frau“ und „Herrn“ geschrieben, akademische Grade dagegen direkt vor den Namen. Da „Professor“ jedoch sowohl eine Berufsbezeichnung als auch ein akademischer Grad sein kann, führte dies immer zu Zweifeln. Jetzt wurde klargestellt, dass wegen dieser Problematik „Prof.“ immer unmittelbar vor dem Namen geschrieben werden sollte.

Bis auf das sogenannte „Professorenproblem“ sind alle diese Neuerungen der leichteren elektronischen Verarbeitbarkeit geschuldet. Die Adresse tritt hinter maschinenlesbare Vermerke zurück. Eine Größe von 8 Punkt ist auch bei Sans-Serif-Schriften schlecht lesbar. Übliche Zeitungsgrundschriften sind 1 Cicero groß (entspricht 12 Punkt). Serifenschriften sind in der Regel leichter lesbar, weshalb der Fließtext der meisten Publikationen mit solchen Schriften gedruckt ist, während serifenlose Schriften für Überschriften verwendet werden. OCR-Programme wiederum erkennen – gerade bei kleinen Schriftarten – serifenlose Schriften besser.

Die Änderungen in der DIN 5008 erleichtern Postzustellungsunternehmen ihre Arbeit. Die nach wie vor manuell erfolgende Endzustellung beim Empfänger (Postverteilung in Unternehmen) dürfte durch die Neuerungen eher erschwert sein.

1.1.2 Schreibwirklichkeit

Für die automatische Erkennung von Adressen in Texten ist es zwar hilfreich, die Normen und Konventionen zu kennen, aber bei weitem nicht hinreichend. Geschäftsbriefe werden immer öfter nicht von professionellen Schreibkräften angefertigt, sondern von Laien, meist mit einem nicht auf Briefe spezialisierten Textverarbeitungsprogramm (etwa „Microsoft Word“, dessen Rechtschreibprüfung als Referenz für die Korrektheit der Adreßschreibung dient). Man kann sich also keineswegs darauf verlassen, daß Anschriften konventions- oder gar normgemäß formatiert sind oder auch nur der korrekten Rechtschreibung entsprechen.

Nicht jede Adresse ist eine Anschrift. Adressen können im Fließtext stehen, mit beispielsweise durch Komma getrennten Bestandteilen, oder mit der Adresse in Klammern

hinter dem Namen. Anschriftfelder enthalten häufig eine klein dargestellte Absenderadresse in einer Zeile mit grafischen Trennern wie · oder *. Adressenlisten können in Tabellenform wiedergegeben sein. All diese Möglichkeiten muß eine automatische Adreßerkennung berücksichtigen, ebenso wie typische Rechtschreibfehler.

1.2 Ortsangaben

Wie bereits gesagt, bestehen Ortsangaben seit 1993 aus einer fünfstelligen Postleitzahl (eventuell mit Länderkennung plus Bindestrich davor, also „D-“[3]) in Verbindung mit einem Ortsnamen, der durch einen Leerschritt davon getrennt ist. Theoretisch gäbe es also 99.999 verschiedene Postleitzahlen, aber nicht alle davon sind vergeben. Die Verknüpfung Postleitzahl/Ort ist in beide Richtungen nicht eindeutig. Es gibt Orte, die wegen der Größe der Zustellbezirke mehrere Postleitzahlen haben, dazu kommen spezielle Postleitzahlen für Großempfänger (Behörden, Kasernen, große Firmen etc.) und Postfachadressen. Andererseits kommt es auch vor, daß mehrere kleine Orte sich eine Postleitzahl teilen. Eine weitgehend vollständige Liste, die dem Programm beigefügt ist, enthält ca. 40.600 Postleitzahl-Ort-Kombinationen bei ca. 29.600 verschiedenen Postleitzahlen und ca. 16.000 verschiedenen Ortsnamen.

Ortsnamen können aus einem Wort bestehen (München), aus einer Kombination von Ort und Ortsteil (München-Bogenhausen), aus mehreren durchgegliederten Ortsnamen, die meist durch Zusammenwachsen mehrerer Orte entstanden sind (Oer-Erkenschwick, Garmisch-Partenkirchen, aber auch: Neu-Ulm), oder Mehrwortlexeme sein. Handelt es sich um Letzteres, gibt es verschiedene Arten von Zusätzen zum eigentlichen Ortsnamen:

geografische Neustadt an der Donau, Freiburg im Breisgau, Rothenburg ob der Tauber, Au in der Hallertau, München bei Bad Berka, Frankfurt (Oder), Lübbenau/Spreewald, Feldberger Seenlandschaft

geschichtliche Königs Wusterhausen, Lutherstadt Wittenberg, Sankt Augustin (hier hat der gesamte Ortsname einen geschichtlichen Bezug)

statusbezeichnende Bad Gögging, Markt Schwaben, Luftkurort Lückendorf

sonstige Groß Kienitz, Alt Rehse

Die Grammatik für Ortsangaben:

`Ortsangabe` → `Postleitzahl Ortsname`

`Postleitzahl` → `(D-)?\d{5}`

`Ortsname` → `Ort`

[3] Die Deutsche Post empfiehlt, von Länderkennzeichen vor Postleitzahlen abzusehen.

`Ortsname` → `Ort-Ortsteil`

`Ortsname` → `Ort-Ort`

`Ortsname` → `Zusatzbezeichner Ort`

`Ortsname` → `Ort Geophrase`

`Geophrase` → `(Geo)`

`Geophrase` → `/Geo`

`Geophrase` → `Präposition Determinator Geo`

Bei Doppel-Orten wie Garmisch-Partenkirchen kann es sein, daß in der Adreßangabe nur einer der beiden Orte genannt wird. Zusatzbezeichnungen wie „Luftkurort“ oder auch „Lutherstadt“ sind optional. Andere, die zur genaueren Identifikation des gemeinten Ortes dienen, können entfallen, wenn der Bezug aus dem Kontext hervorgeht – wer in Abensberg auf „Neustadt“ verweist, meint den Nachbarort Neustadt an der Donau oder drückt sich unklar aus. Wegen der Bedeutung als Börsen- und Messestandort und der Größe der Stadt ist es wahrscheinlicher, daß „Frankfurt“ ohne Zusatz für Frankfurt am Main steht als für Frankfurt/Oder. Da Adressen ohnehin die Postleitzahl als zusätzlichen Indikator bieten, ist es nicht selten, daß die Zusätze nicht mit angegeben werden.

Die Zuordnung von Zusätzen zu Orten ist, von Rechtschreibvarianten abgesehen, eindeutig. Rothenburg ist mit vollem Namen immer „Rothenburg ob der Tauber“, „Rothenburg an der Tauber“ ist falsch. Die passenden Zusätze müssen einem Lexikon entnommen werden.

1.3 Straßennamen

Unter Straßenangaben werden hier auch Postfachangaben subsumiert, da sie in Adreßangaben die gleiche Position einnehmen.

1.3.1 Rechtschreibung

Straßennamen bereiten die größten Schwierigkeiten in der Rechtschreibung; bei Ortsnamen oder Eigennamen sind Fehler wesentlich seltener. Seit der Rechtschreibreform von 1996 ist die Schreibung „Strasse“ häufiger als früher anzutreffen, was auf den Glauben an die Abschaffung des „ß“ oder eine größere Unsicherheit in bezug auf die „ss/ß“-Schreibung zurückzuführen ist. An der Schreibung von Straßennamen hat sich durch die Reform nichts geändert. Die wichtigsten Regeln [Duden, R189 ff.]:

- Das erste Wort eines Straßennamens wird groß geschrieben, ebenso alle zum Namen gehörenden Adjektive und Zahlwörter.
  Im Trutz, Am Alten Lindenbaum, Kleine Bockenheimer Straße, An den Drei Tannen

- Straßennamen, die aus einem einfachen oder zusammengesetzten Substantiv (auch Namen) und einem für Straßennamen typischen Grundwort bestehen, werden zusammengeschrieben.
  Schloßstraße, Brunnenweg, Bismarckring, Augustaanlage, Wittelsbacherbrücke, Marienwerderstraße, Stresemannplatz

- Soll in einem Straßennamen ein altes Besitzverhältnis ausgedrückt werden, tritt oft ein Genitiv-s auf. In solchen Fällen ist gelegentlich auch Getrenntschreibung möglich.
  Brandtstwiete, Oswaldsgarten; Graffelsmanns Kamp

- Straßennamen, die aus einem ungebeugten Adjektiv und einem Grundwort zusammengesetzt sind, werden zusammengeschrieben.
  Altmarkt, Neumarkt, Hochstraße

- Getrennt schreibt man dagegen, wenn das Adjektiv gebeugt ist.
  Große Bleiche, Langer Graben, Neue Kräme, Französische Straße

- Getrennt schreibt man auch bei Ableitungen auf -er von Orts- und Ländernamen.
  Münchener Straße, Bad Nauheimer Weg, Am Saarbrücker Tor, Schweizer Platz, Herner Weg, Kalk-Deutzer Straße

- Bei Ortsnamen, Völker- oder Familiennamen auf -er wird jedoch [...] zusammengeschrieben.
  Drusweilerweg, Römerplatz, Herderstraße

- Mehrteilige Straßennamen werden gebeugt.
  Er wohnt in der Oberen Riedstraße

- Den Bindestrich setzt man, wenn die Bestimmung zum Grundwort aus mehreren Wörtern besteht.
  Albrecht-Dürer-Allee, Paul-von-Hindenburg-Platz, Kaiser-Friedrich-Ring, Ernst-Ludwig-Kirchner-Straße, E.-T.-A.-Hoffmann-Straße, Professor-Sauerbruch-Straße, Van-Dyck-Straße, Berliner-Tor-Platz, Bad-Kissingen-Straße, Sankt-Blasien-Straße, Am St.-Georgs-Kirchhof, Bürgermeister-Dr.-Meier-Platz, Von-Repkow-Platz, v.-Repkow-Platz

Schwierigkeiten treten vor allem bei der Getrennt- und Zusammenschreibung und dem korrekten Durchgliedern mit Bindestrich auf. So wird aus der Münchener Straße häufig die Münchenerstraße, der Gerhart-Hauptmann-Ring zum Gerhart-Hauptmannring oder die E.-T.-A.-Hoffmann-Straße gar zur E T A Hoffmannstraße, während die Wittelsbacherbrücke nicht mehr dem Geschlecht der Wittelsbacher, sondern als Wittelsbacher Brücke einem imaginären Ort Wittelsbach gewidmet wird. Solche Varianten sollten nach Möglichkeit erkannt werden, da diese Formen der Falschschreibung häufig sind. Wenn die falsch geschriebenen Varianten auf die korrekte zurückgeführt werden können, ermöglicht

dies sogar eine automatische Richtigstellung. Nicht in jedem Fall ist es jedoch zweifelsfrei möglich, denn die Nürnbergerstraße kann auch auf den Nachnamen „Nürnberger“ zurückzuführen und somit korrekt sein.

1.3.2 Grammatik für Straßen

(a) `Straße` → `Postfachbezeichner Postfachnummer`

(b) `Postfachnummer` → `(\d|␣){1,8}`

(c) `Postfachbezeichner` → *Postfach, Pf., Postf., PF*

(d) `Straße` → `Straßenname Hausnummer`

(e) `Hausnummer` → `^\d{1,3}\D?(-\d{1,3}\D?)?(/.*)?$`

(f) `Straßenname` → `Name+Straßengrundwort`

(g) `Straßengrundwort` → *Straße, Ring, Platz, Allee, Chaussee, Brücke, Damm, Gasse, Graben, Markt, Platz, Promenade, Ring, Steg, Tor, Ufer, Weg, Twiete, Leite, Kamp, Deich*

(h) `Straßenname` → `Geo+`*er* `Straßengrundwort`

(i) `Straßenname` → `Vorname-Nachname-Straßengrundwort`

(j) `Straßenname` → `Titel-Vorname-Nachname-Straßengrundwort`

(k) `Straßenname` → `Titel-Nachname-Straßengrundwort`

(l) `Straßenname` → `Straßengrundwort Determinator Nominalphrase`

(m) `Straßenname` → `Lokalpräposition Nominalphrase`

(n) `Straßenname` → *Alte, Neue, Große, Kleine, Vordere, Hintere, Obere, Untere, Innere, Mittlere, Äußere, Westliche, Östliche, Südliche, Nördliche, Lange, Kurze* `Straßenname`

(o) `Straßenname` → `SonstigerStraßenname`

Die letzte Regel kann nicht leicht näher spezifiziert werden. Die meisten Straßennamen beinhalten ein Straßengrundwort am Ende und sind in Regeln faßbar. Schwierig wird es bei Fällen, in denen das Grundwort am Anfang steht und von einer beliebigen Nominalphrase gefolgt wird: *Straße des Kindes, Platz der Opfer des Nationalsozialismus* (Regel l), oder wenn gar kein Straßengrundwort vorhanden ist: *Tüünlüüd, Abgunst*. Solche Straßennamen sind in der Schwammregel o zusammengefaßt.

Da diese Regeln so unspezifisch sind, ist es zur Erkennung von Straßennamen nötig, auf ein möglichst vollständiges Lexikon zurückzugreifen, wenn man eine große Präzision erreichen will. Ohne ein solches Lexikon hat man die Wahl, sehr unvollständig zu erkennen, oder eine schlechte Präzision – viele falsche Treffer – in Kauf zu nehmen. Im Rahmen dieser Arbeit wurde ein Lexikon von deutschen Straßennamen aus am CIS vorhandenen Daten erstellt, das zur Zeit 212.000 Einträge umfaßt.

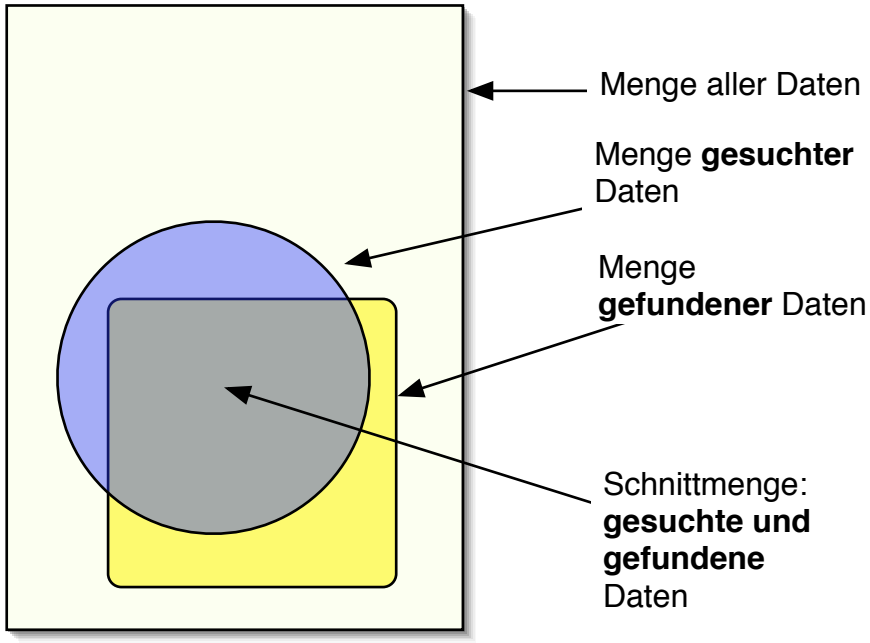

Abbildung 1.1: Präzision und Recall

$$\text{Präzision} = \frac{\text{gesucht} \cap \text{gefunden}}{\text{gefunden}} \qquad \text{Recall} = \frac{\text{gesucht} \cap \text{gefunden}}{\text{gesucht}}$$

1.4 Eigennamen

Die Grammatik von Eigennamen soll hier nicht weiter thematisiert werden. Es ist ein so großes und schwer faßbares Gebiet, daß es allein Thema umfangreicher Dissertationen ist. Bislang kann es trotz aller Anstrengungen als ungelöstes Problem der Computerlinguistik gelten, Eigennamen annähernd vollständig und fehlerfrei zu erkennen.

1.5 weitere Adreßbestandteile

1.5.1 Telefon- und Faxnummern

Telefon- und Faxnummern sind bei vielen Adressen mit angegeben. Da ihre Syntax völlig identisch ist, können sie nur durch explizite Auszeichnung unterschieden werden.

`Telefon` → `Telefonnummernbezeichner Telefonnummer`

`Fax` → `Faxnummernbezeichner Telefonnummer`

`Telefonnummernbezeichner` → *Telefonnummer, Telefon, Tel., Tel, Fon*

`Faxnummernbezeichner` → `Faxnummer, Fax, Telefax`

`Telefonnummer` → `Vorwahl Rufnummer`

`Vorwahl` → `Länderkennung Ortsnetzkennung`

`Länderkennung` → `(\+49|0049)`

`Ortsnetzkennung` → `0?[1-9]\d{1,4}`

`Rufnummer` → `[1-9]\d{2,7}`

`Rufnummer` → `Anschlußpräfix Durchwahl`

`Rufnummer` → `[1-9]\d{2,6}`

`Durchwahl` → `Computerzusatzdurchwahl Endanschluß`

`Computerzusatzdurchwahl` → `\d{1,3}`

`Endanschluß` → `(0|[1-9]\d{0,5}`

Die einzelnen Bestandteile der Nummern können durch Klammern[4] als optional gekennzeichnet sein, und sind in der Regel durch Leerzeichen, Bindestrich oder Schrägstrich getrennt. Leerzeichen können auch die einzelnen Nummernbestandteile selbst untergliedern – normgerecht wäre dies in Paaren, von hinten beginnend, bei ungerader Zahl mit einer Dreiergruppe am Anfang oder einer einzelnen Ziffer (seltener). Dies ist in den oben angegebenen regulären Ausdrücken aus Gründen der Übersichtlichkeit nicht berücksichtigt. Die Nummernerkennung in Clark arbeitet nicht auf Basis regulärer Ausdrücke, da diese auch die Leerzeichen berücksichtigen müßten und entsprechend lang und langsam wären, ohne besser zu erkennen als die gewählte Lösung, die in Kapitel 3.2 beschrieben ist.

Es wäre denkbar und auch sinnvoll, gefundene Telefon- und Faxnummern in ein einheitliches Format zu bringen. Dazu braucht man eine komplette Liste deutscher Vorwahlnummern, um feststellen zu können, an welcher Stelle die Trennung von Vorwahl und

[4] Da zu einer öffnenden Klammer zwingend eine schließende gehört, was den Ausdruck nicht mehr regulär sein läßt, wird ein regulärer Ausdruck, der dies berücksichtigt, übermäßig lang, da er zu jeder nicht geklammerten Alternative die gleiche mit Klammern haben muß. Andere Formen des Parsings sind deshalb sinnvoller.

Endanschluß vorgenommen werden muß. Vorwahlnummern (Ortsnetzkennungen) sind mit führender Null mindestens drei- und höchstens sechsstellig. Die komplette Rufnummer ist wegen hauseigener Telefonanlagen quasi beliebig lang, wobei 16 Ziffern inklusive Länderkennung selten überschritten werden.

Deutsche Mobilfunknummern sind zur Zeit mit Länderkennung exakt zwölfstellig, genauso wie die neuen ortsunabhängigen 0700er-Rufnummern. Sonderrufnummern wie gebührenfreie (0800, 0130) oder mit besonderen Entgelten beaufschlagte (0900, 0190) sind in der Regel zehnstellig.

1.5.2 E-Mail

Im Unterschied zu den bisher besprochenen Adreßangaben entstammt die E-Mail-Adresse dem Internetzeitalter, ist für die Kommunikation mittels Rechnersystemen konzipiert worden und sollte dementsprechend einfach von einem Rechnersystem erkannt werden können.

Die erste E-Mail wurde 1972 verschickt. Das dazu notwendige E-Mail-Programm war von Ray Tomlinson, Mitarbeiter bei BBN, der Firma, die das Arpanet, den Vorläufer des Internet, im Auftrag der Arpa[5] in Betrieb brachte, entwickelt worden. Zu dieser Zeit waren gerade einmal 23 Hosts im Arpanet verbunden – mit einer Geschwindigkeit von 50 Kilobit pro Sekunde, die ziemlich genau dem entspricht, was heute ein analoges Modem bereitstellt, mithin die langsamste Methode, ins Internet zu kommen.

Im Internet verwendete Protokolle sind in RFCs[6] quasi-standardisiert. Der erste Vorschlag für ein standardisiertes E-Mail-Format ist in RFC724 – „Proposed official standard for the format of ARPA Network messages“ – vom 12. Mai 1977 dargelegt. Dieser wurde im November desselben Jahres vom „Standard for the format of ARPA network text messages“ (RFC733) abgelöst. Der Nachfolger, im August 1982 veröffentlicht, hieß „Standard for the format of ARPA Internet text messages“ (RFC822 bzw. STD11).

Jeffrey Friedl entwickelt in der 1998er Ausgabe seines Buches „Mastering Regular Expressions“ – eines Standardwerks zu dem Thema – einen regulären Ausdruck für syntaktisch korrekte E-Mail-Adressen, der 107 Zeilen Perl-Code umfaßt (ohne Kommentare und Leerzeilen). Das Codebeispiel ist online abrufbar unter [Friedl1], und es ist deshalb so lang, weil es RFC822 abdeckt. 1982 gab es im Internet weniger als 500 Hostrechner, und es waren E-Mail-Adressen auch folgender Form erlaubt [RFC822, 6]:

```
":sysmail"@  Some-Group. Some-Org,
Muhammed.(I am  the greatest) Ali @(the)Vegas.WBA
```

[5] Advanced Research Projects Agency, ein zunächst dem US-Verteidigungsministerium unterstelltes Institut, gegründet 1957, das dem Zweck diente, nach dem „Sputnik-Schock“ den USA wieder die Führerschaft in Sachen militärisch verwertbarer Wissenschaft und Technik zu bringen. Dort wurde die Idee eines dezentralen Computernetzwerks geboren.

[6] Request for Comment, siehe <URL:http://rfc-editor.org/>

RFC2822 löste im Jahr 2001 RFC822 ab und definiert das Format von E-Mail-Adressen deutlich strikter. Die exakte Spezifikation ist unter [RFC2822] zu finden. In „Mastering Regular Expressions, 2nd Edition“ faßt Friedl sich deshalb auch wesentlich kürzer [Friedl2, 72f.]. Aus seiner Beschreibung ergibt sich zusammengefaßt diese Grammatik:

```
E-Mail-Adresse →
\w[-.\w]*\@[-a-z0-9]+(\.[-a-z0-9]+)*\.
(com|edu|gov|int|mil|net|org|biz|info|name|museum|coop|aero|[a-z][a-z])
```

1.6 Adreßumfeld

Aus einigen der oben besprochenen Adreßbestandteile war bereits ersichtlich, daß das Umfeld wertvolle Hinweise zur Erkennung liefern kann – etwa bei Telefon- und Faxnummern, die als solche durch einen vorstehenden Bezeichner qualifiziert werden müssen. Für die gesamte Adresse kann es ebenso semantische oder strukturelle Qualifikatoren geben.

Häufig werden Adreßblöcke im Fließtext durch bestimmte Phrasen eingeleitet, zum Beispiel „Anschrift:“, „Adresse“, „Kontakt“, „Hier finden Sie uns“, oder mit Funktionsangabe: „Lieferadresse“, „Redaktion“, „Verwaltung“ etc.

Solche einleitenden Begriffe können dazu dienen, den oberen Abschluß einer Adresse zu finden. Die eine Adresse abschließenden Bestandteile wie Ort, Telefonnummer oder E-Mail-Adresse sind relativ leicht zu erkennen, aber der Beginn eines Adreßblocks ist zumeist der Name des Adressaten. Wenn man die restlichen Teile einer Adresse erkannt hat, ist es so möglich, den Namen dadurch mit einer gewissen Wahrscheinlichkeit herauszufinden, daß er den Platz zwischen dem Rest der Adresse und der die Adresse ankündigenden Phrase einnimmt. Die Namenserkennung in Clark basiert zum Teil auf dieser Annahme.

Je nach Textart, auf der die Erkennung stattfindet, kann datei- bzw. dokumenttypimmanente Struktur zur Erkennung herangezogen werden. Bei elektronisch erstellten Briefen, die auf derselben Vorlage basieren, könnte das Anschriftfeld durch Formatierungsanweisungen, strukturelles Markup oder die Position in der Datei identifiziert beziehungsweise eingegrenzt werden.

Wenn es sich bei den zu durchsuchenden Dateien um Webseiten im HTML-Format handelt, ist die Überprüfung auf hilfreiche Strukturen deshalb eher simpel, weil es sich bei HTML um ein menschenlesbares Format mit überschaubarer Anzahl an Strukturelementen handelt.

1.6.1 HTML – die Hypertext Markup Language

HTML ist eine Textauszeichnungssprache, das heißt der eigentliche Text ist mit strukturierenden Auszeichnungselementen – „Tags“ – versehen, die dem ihn einlesenden oder

darstellenden Programm Hinweise zur Verarbeitung geben. Dabei ist zwischen spezifischem bzw. layoutorientiertem und generalisiertem bzw. strukturierendem Markup zu unterscheiden. Einfach ausgdedrückt: spezifisches Markup gibt an, daß ein Textstück in „Verdana, 18 Punkt, fett“ darzustellen ist, während generalisiertes Markup ein Textstück als „Überschrift erster Ordnung“ identifiziert und somit einen semantischen Hinweis auf die Art des Inhalts gibt, wobei die Art der Darstellung unspezifiziert bleibt.

Bereits die erste Version von HTML war eine Mischform von spezifischem und generalisiertem Markup. Die meisten Tags dienten der Strukturierung: `H2` (Überschrift zweiter Ordnung), `ADDRESS` (Adresse), `MENU` (Navigationsbereich) etc. Bald darauf gab es aber aber auch Tags wie `B` (fette Darstellung) oder `I` (kursiv).

Bis zur Version 3.2 des HTML-Standards wurde der Fokus immer stärker auf spezifisches Markup gelegt, nicht zuletzt deshalb, weil der Standard proprietären Erweiterungen der Browserproduzenten Netscape und Microsoft beigebogen wurde, die HTML um Tags wie `FONT` (Schrifttyp, -größe, -schnitt), `BLINK` (blinkender Text) oder `MARQUEE` (Laufschrift) erweiterten. Für pixelgenaues Design wird auch heute noch vielfach die Verschachtelung von `TABLE`-Strukturen verwendet, die dafür eigentlich nie gedacht waren. Erst ab Version 4.0 des HTML-Standards (1997) wurde auf eine striktere Trennung von Form und Inhalt Wert gelegt. Die aktuelle Version, XHTML 1.0, sieht generalisiertes Markup plus Formatierungsanweisungen in Stylesheets vor, die angeben, welches Layout mit welchen Tags zu verwenden ist.

Das heutige Web ist eine bunte Mischung aus allem von HTML 3.2 bis XHTML 1.0, die wenigsten Webseiten halten irgendeinen der Standards korrekt ein. Laut einer Untersuchung aus dem Jahr 2001 enthalten 0,71% aller Webseiten valides HTML. Mehr als 99 Prozent aller Seiten sind syntaktisch inkorrekt [Parnas, 80]. Das ist aber nur eine Seite des Problems. Die andere liegt in semantischer Unkorrektheit [Parnas, 37]:

```
<P>And then the big bad wolf said<P>
<H1>Then I'll huff and I'll puff and I'll blow your house down </H1>
```

Listing 1.1: syntaktisch korrektes, semantisch falsches HTML

Der erste Satz ist hier als normaler Absatz ausgezeichnet, der zweite als Überschrift erster Ordnung. Der Zweck ist, den zweiten Satz viel größer darzustellen als den ersten, und das funktioniert auch, sofern man die Seite mit einem Grafikbrowser betrachtet (Abbildung 1.2).

Problematisch wird es, wenn ein Rechner den Text verarbeitet und in diesem Fall nicht nur keine semantische Information bekommt, sondern sogar eine falsche. Genauso problematisch ist eine solche Art der Auszeichnung, wenn die Seite von einem Menschen in unüblicher Weise betrachtet wird – etwa von einem Sehbehinderten, der sie sich von einem Screenreader vorlesen läßt. Der Screenreader „sieht“ das gleiche, was auch ein Rechner sieht; er wird dem Menschen mitteilen, daß in der Hauptüberschrift der Seite gehufft und gepufft wird.

Neben der Trennung von Struktur und Layout wird seit einiger Zeit das „Semantic Web“ als Möglichkeit propagiert, Computern die Erfassung des Inhalts von Webseiten zu erleichtern[7]. Ein System einfacher Ontologien soll dabei den Wissensaustausch automatisieren und vereinfachen helfen. Vor dem Hintergrund, daß schon das simple HTML nicht richtig angewendet wird, ist es fraglich, ob die wesentlich komplizierteren Schemata des Semantic Web, die es erfordern, Informationen genau zu spezifizieren und kategorisieren, tatsächlich real umgesetzt werden können.

1.6.2 Adressen in der Markup-Suppe

Für Adressen gibt es ein eigenes Tag, das `address`-Tag (in XHTML sind in Anlehnung an XML[8] nur noch kleingeschriebene Tags erlaubt). Dieser Arbeit liegt ein Korpus von 119 Webseiten[9] bei, von denen jede eine oder mehrere Adressen enthält. Das `address`-Tag wird in einer einzigen davon – immerhin tatsächlich für eine Adresse – benutzt.

Kann man das die Adresse umgebende Markup trotzdem für die Erkennung der Adresse heranziehen?

Es gibt Programme, die beliebiges Markup dazu benutzen, strukturierte Informationen aus Webseiten automatisch zu extrahieren: sogenannte Wrapper-Generatoren [Ziegler, 86]. Brauchbar sind diese vor allem dann, wenn eine große Menge von immer gleich strukturierten Daten ausgelesen werden soll. Ein möglicher Anwendungsbereich sind dynamisch generierte Seiten, die bestimmte Datensätze aus einer Datenbank holen und in HTML für die Bildschirmdarstellung verpacken. Mit einigen Beispielen ist es einem Wrapper-Generator wie Wien, Stalker oder Lixto möglich, eine virtuelle Schablone über die Seite zu legen und so die gewünschten Informationen zu extrahieren. Ändert sich der Aufbau der Seite, muß die Schablone wieder angepaßt werden.

Listings 1.2, 1.3 und 1.4 zeigen den Adreßteil von drei wahllos aus dem Korpus herausgegriffenen Dateien im Quelltext. Bis auf Zeilenumbrüche wurde an dem Code nichts

[7] siehe <URL:http://w3.org/2001/sw/>
[8] eXtensible Markup Language, eine Meta-Markup-Sprache, um Markupsprachen zu definieren
[9] Das Korpus besteht aus Webseiten aus ganz unterschiedlichen Bereichen, von der privaten Homepage bis zum Firmenimpressum. Es wurde im Dezember 2003 erstellt.

Abbildung 1.2: Screenshot einer HTML-Datei mit dem oben gezeigten Fragment

```
<P ALIGN="CENTER">
</B>
<A HREF="./Grosse_Grossen/Karte/karte.html">
<B><FONT SIZE="+1">Blasewitzer Strasse 78,  01307 Dresden</FONT>
    </B></A>
<B><FONT SIZE="+1"></P>
<P ALIGN="CENTER">Tel./Fax 0351- 4 42 53 42</FONT></B>
```

Listing 1.2: HTML um eine Adresse, Beispiel 1

```
<tr valign="middle">
  <td valign="top">
     <div align="left"><font face="Arial, Helvetica, sans-serif" size
         ="2"><br>
       Bon Prix Handelsgesellschaft mbH<br>
       Haldesdorfer Stra&szlig;e 61</font> <font face="Arial,
           Helvetica, sans-serif" size="2"><br>
       22179 Hamburg<br>
      </font><font face="Arial, Helvetica, sans-serif" size="2"><br>
       Telefon: 040 - 64 62 - 0<br>
      E-Mail: service@bonprix.de</font> <br>
      <br>
    </div>
  </td>
</tr>
```

Listing 1.3: HTML um eine Adresse, Beispiel 2

```
<p>
  <B>Firmenwelt GmbH</B><BR>
  Grundelbachstra&szlig;e 112B<BR>
  69469 Weinheim<BR>
</p>
<p>
  Postfach 20 01 40<BR>
  69459 Weinheim
</p>
<p>
  Telefon: +49 6201 185525<BR>
  Telefax: +49 6201 183984<BR>
  E-Mail: <A HREF="mailto:info@firmenwelt.de"> info@firmenwelt.de</A>
      <BR>
  Homepage: www.firmenwelt.de
</p>
```

Listing 1.4: HTML um eine Adresse, Beispiel 3

verändert. In Listing 1.2 befindet sich die Adreßangabe in zwei getrennten, zentrierten Absätzen. Der Standard ist HTML 3.2, wobei das schließende Tag in Zeile 2 kein öffnendes hat. Listing 1.3 entspricht HTML 4.x Transitional und verwendet neben Tabellen bereits Stylesheets zur Formatierung. Das `div`-Tag ist das semantisch leerste, das in HTML definiert ist. Es bezeichnet einen Container beliebigen Inhalts. Listing 1.4 dürfte ebenfalls HTML 4.x Transitional sein. Hier sind die verschiedenen Adreßbestandteile auf drei Absätze verteilt.

Die drei Listings zeigen das Dilemma: Sie sind völlig unterschiedlich. Das Markup, das in verschiedenen Websites für Adressen verwendet wird, kann nicht auf andere übertragen werden, um dort ebenfalls Adressen zu finden.

Listing 1.5 zeigt einen Teil einer Adressenliste. Bei Listen, die sich auf einer Seite oder einem Site befinden, sieht es anders aus. Die beiden Adressen sind identisch ausgezeichnet.

In Listing 1.6 wurden die Tag-Attribute entfernt, die Adreßangaben durch Platzhalter ersetzt und die Nicht-Adreß-Blöcke entfernt, um die Übereinstimmungen besser sehen zu können. Das Beispiel zeigt deutlich, daß das Tag-Gerüst bei Adressenlisten sehr hilfreich sein kann. Wenn mindestens eine, besser zwei Adressen zweifelsfrei erkannt sind, von anderen aber nur Fragmente erkannt wurden, können die Teiladressen mit Hilfe der umgebenden Strukturinformation vervollständigt werden.

```
<tr>
  <td colspan="2" align="left" valign="bottom"><p align="center">
  <font face="Arial">
  <b>Produktpalette:
  </b>Gemüsesaatgut <br>
  <b>Serviceleistungen:</b> Preisliste des Gemüsesaatgutes wird auf
      Anfrage zugeschickt. <br>
  Ab November 1999 Katalog und Preisliste auch für Spanien, Italien,
      Frankreich, Holland,
  England.</font></td></tr>
<tr valign="bottom"><td colspan="3" align="right"><p align="left">&
    nbsp;</td></tr>
<tr>
  <td bgcolor="#3366FF" rowspan="2" align="center" valign="middle"><
      font face="Arial"
  size="+1" color="#FFFFFF"><b>35</b></font></td>
  <td align="left" valign="bottom"><font face="Arial">Wilhelm Schoell
       Saaten GmbH <br>
  Im Kalten Brunnen 14 <br>
  72666 Neckartailfingen</font></td>
  <td align="left" valign="bottom"><font face="Arial"><b>AP: </b>
      Bernd Hammer <br>
  <b>Tel.:</b> 07127/ 220 13 <br>
  <b>Fax: </b>07127/ 228 16</font></td>
</tr>
<tr>
  <td colspan="2" align="left" valign="bottom"><p align="center"><
      font face="Arial"><b>Produktpalette:
  </b>Steckzwiebeln; Gründüngung (s. Ackerbauliste) <br>
  <b>Serviceleistungen: </b>bundesweite Lieferung</font></td>
</tr>
<tr valign="bottom">
  <td colspan="3" align="right"> </td>
</tr>
<tr>
  <td bgcolor="#3366FF" rowspan="2" align="center" valign="middle">
  <font face="Arial" size="+1" color="#FFFFFF"><b>49</b></font></td>
  <td align="left" valign="bottom"><font face="Arial">Rijk Zwaan GmbH
       <br>
  Werler Str. 1 <br>
  59514 Welver</font></td>
  <td align="left" valign="bottom"><font face="Arial"><b>AP: </b>
      Edgar Schweizer <br>
  <b>Tel.: </b>02384 / 501 141 <br>
  <b>Fax: </b>02384 / 501 110 o. -178</font></td>
</tr>
```

Listing 1.5: eine Adressenliste

```
<tr>
  <td><font><b>35</b></font></td>
  <td><font>Firmenname <br>
  Straße <br>
  PLZ Ort</font></td>
  <td><font><b>AP: </b>Ansprechpartner <br>
  <b>Tel.: </b>Telefonnummer<br>
  <b>Fax: </b>Faxnummer</font></td>
</tr>
<tr>
  <td><font><b>49</b></font></td>
  <td><font>Firmenname <br>
  Straße<br>
  PLZ Ort</font></td>
  <td><font><b>AP: </b>Ansprechpartner <br>
  <b>Tel.: </b>Telefonnummer <br>
  <b>Fax: </b>Faxnummer</font></td>
</tr>
<tr>
```

Listing 1.6: Adressenliste bereinigt

2 Kritische Würdigung der Magisterarbeit „Maschineller Adressen-Abgleich“ von Jürgen Altfeld

Bereits im Jahr 2000 entstand am CIS eine Magisterarbeit, die Adreßerkennung in Webseiten zum Thema hatte: „Maschineller Adressen-Abgleich – Extraktion und Struktur-Analyse von Adressen aus Web-Seiten“ von Jürgen Altfeld. Da jene Arbeit eine ähnliche (wenn auch nicht die gleiche) Zielsetzung wie diese, aber einen stark unterschiedlichen Ansatz wählt, soll im Folgenden näher auf einige Punkte eingegangen werden.

2.1 Ansätze

Altfeld legt folgende Vorgehensweise zur Adreßextraktion zugrunde [Altfeld, 6]:

- Umwandlung der Web-Seite in reinen ASCII-Text[1]
- Extraktion der Stellen im ASCII-Text, in denen Adressen vermutet werden
- Genaue Erkennung der Adressen und ihrer Bestandteile mit wörterbuchgestütztem Tagging und regulären Ausdrücken

Bestandteil der Arbeit ist ein Prototyp eines Adreßerkenners, der obige Vorgehensweise für ein kleines Testset an Webseiten umsetzt.

2.2 Funktionsweise

Der Prototyp ist in Java implementiert, in Version 1.2 von Sun Microsystems, und läuft als Browser-Applet auf der Windows-Plattform.

[1] Gemeint ist hier Plain Text oder Nur-Text, also Text ohne Formatierungsanweisungen, wie er mit einer mechanischen Schreibmaschine oder einem Text-Editor wie Notepad erstellt werden kann. ASCII bezeichnet einen Zeichensatz, in dem keine deutschen Sonderzeichen vorhanden sind, und wäre somit für Erkennung deutscher Adressen untauglich. Altfelds Nomenklatur wird in diesem Kapitel beibehalten.

Firma	Wir bieten Ihnen folgende Leistungen:
Gruber Immobilien	- Beratung
Wuchergasse 10	- Verwaltung
80634 München	- Vermittlung

Tabelle 2.1: Beispiel eines Tabellenlayouts

2.2.1 HTML-Konvertierung

Der erste Verarbeitungsschritt ist die Konvertierung des HTML-Quellcodes in ASCII-Text. Der Prototyp ersetzt dabei bestimmte Tags wie `P` oder `H1` durch Zeilenumbrüche, konvertiert HTML-Entitäten wie `&Auml;` in die Sonderzeichen, die sie repräsentieren, und verwirft den Rest des Markup [Altfeld, 8].

Altfeld weist darauf hin, daß es bei der Konvertierung von HTML-Tabellen zur Vermischung von Spalten kommen kann und führt das Beispiel in Tabelle 2.1 – eine zu Layoutzwecken verwendete Tabellenstruktur – an [Altfeld, 11].

Tabellen werden in HTML durch Zeilen definiert, die Zellen enthalten. Obiges Beispiel besteht aus vier Zeilen mit jeweils zwei Zellen. Zellen können auch mehrzeiligen Inhalt haben, siehe Tabelle 2.2.

Firma	Wir bieten Ihnen folgende Leistungen:
Gruber Immobilien	- Beratung
Wuchergasse 10	- Verwaltung
80634 München	- Vermittlung

Tabelle 2.2: Tabellenlayout mit mehrzeiligen Zellen

Programme lesen Text serialisiert ein, also als lange Zeichenkette. Dies trifft natürlich auch auf HTML-Tabellendefinitionen zu. In menschlicher Draufsicht auf einen Text könnte man als sagen „von links nach rechts und von oben nach unten“. Im Fall von Altfelds Beispiel (Tabelle 2.1) bekommt ein Programm oder der Nutzer eines Screenreaders also folgenden Input:

```
Firma
Wir bieten Ihnen folgende Leistungen
Gruber Immobilien
-Beratung
Wuchergasse 10
-Verwaltung
80634 München
-Vermittlung.
```

[Altfeld, 11] Die spaltenweise angegebenen Informationen sind durchmischt.

Im Fall von Tabelle 2.2 ergäbe sich dagegen dieser Input:

```
Firma
Gruber Immobilien
Wuchergasse 10
80634 München
Wir bieten Ihnen folgende Leistungen
-Beratung
-Verwaltung
-Vermittlung
```

Nimmt man an, daß Tabellen wie 2.1 in Webseiten häufig vorkommen, sind daraus zwei mögliche Schlüsse zu ziehen:

- Der serialisierte Input muß in eine Baumstruktur überführt werden, die neben zeilenweiser auch spaltenweise Betrachtung ermöglicht.
- Adreßbestandteile müssen für sich erkannt werden, das heißt man darf nicht einen Matcher schreiben, der komplette Adressen erkennt, sondern eigene für jeden Bestandteil einer Adresse, und baut die Adressen am Ende der Erkennung in einem eigenen Prozeß zusammen.

Clark, der im Rahmen dieser Arbeit erstellte Adreßerkenner, nutzt die zweite Methode, er würde also auch die Adresse aus 2.1 bis auf den Namen korrekt erkennen. Bei entsprechend ausgearbeiteten Routinen zur Namenserkennung könnte die Adresse trotz der Durchmischung komplett erkannt werden.

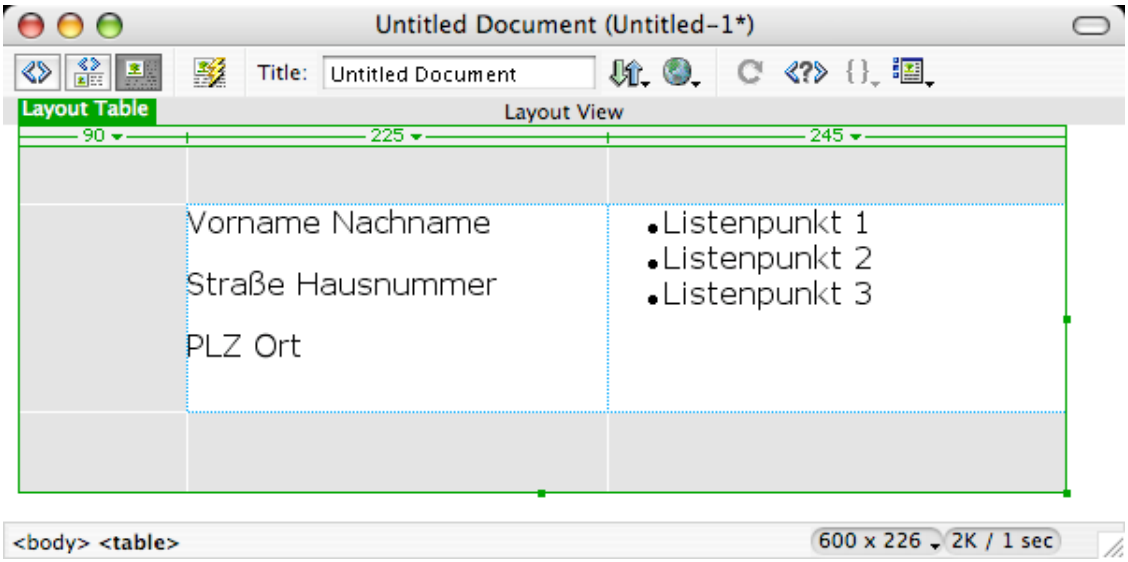

Abbildung 2.1: WYSIWYG-Webseitenerstellung

In praxi sind die Schwierigkeiten mit Tabellen geringer, als man durch das oben angegebene Beispiel vermuten könnte. Tabellenlayouts werden in der Regel mit Wysiwyg[2]-Editoren erstellt. Abbildung 2.1 zeigt ein typisches Beispiel dafür. Hier wurde ein simples Tabellenlayout erstellt.

Was das Bild nur schlecht zeigen kann, ist, daß es sehr unintuitiv wäre, einen zusammengehörenden Block wie eine Adresse auf mehrere Zeilen aufzuteilen. Der Bildschirmschnappschuß zeigt das Programm Macromedia Dreamweaver MX, ein beliebtes Website-Erstellungstool, in Tabellen-Layout-Ansicht. Darin kann man auf einfache Art Blöcke auf der Seite definieren. Diese Blöcke sind Tabellenzellen. Schon die Art der Darstellung zeigt, daß beim Erstellen der Seite in den Hintergrund treten soll, daß man Tabellenzellen definiert. Für die dezidierte Erstellung von Tabellen gibt es eine eigene Funktion, die eine völlig andere Darstellung präsentiert (Abb. 2.2).

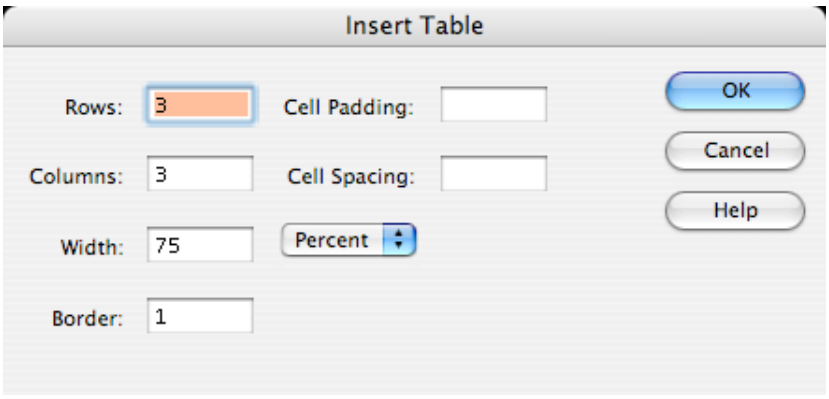

Abbildung 2.2: Erstellen einer Tabelle

Beim Füllen der Tabelle werden die Zellengrößen automatisch angepaßt und der Zelleninhalt gegebenenfalls umbrochen (Abb. 2.3).

Bei meinen Untersuchungen des Aufbaus von Webseiten für diese Arbeit habe ich kein Beispiel wie Tabelle 2.1 gefunden. Das heißt nicht, daß es so etwas nicht geben kann, aber es deutet darauf hin, daß zumindest dieses Problem nicht allzu häufig auftritt.

2.2.2 Suche nach Adreßkandidaten

Altfelds Programm sucht mit einem langen regulären Ausdruck nach Schlüsselwörtern [Altfeld, 15f.]. Falls in einem Segment mehrere Schlüsselwörter gefunden werden, wird dieses Segment als eine Adresse enthaltend ausgezeichnet. Der reguläre Ausdruck enthält einen Ausdruck für fünfstellige Zahlen (Postleitzahlen) und eine Menge von Wörtern, die in Adressen vorkommen können.

[2]What You See Is What You Get, während der Erstellung soll man bereits das gewünschte Endergebnis sehen können, ähnlich der Layout-Ansicht in Microsoft Word – im Unterschied zu Quelltext-Editoren, mit denen man HTML-Code erstellt.

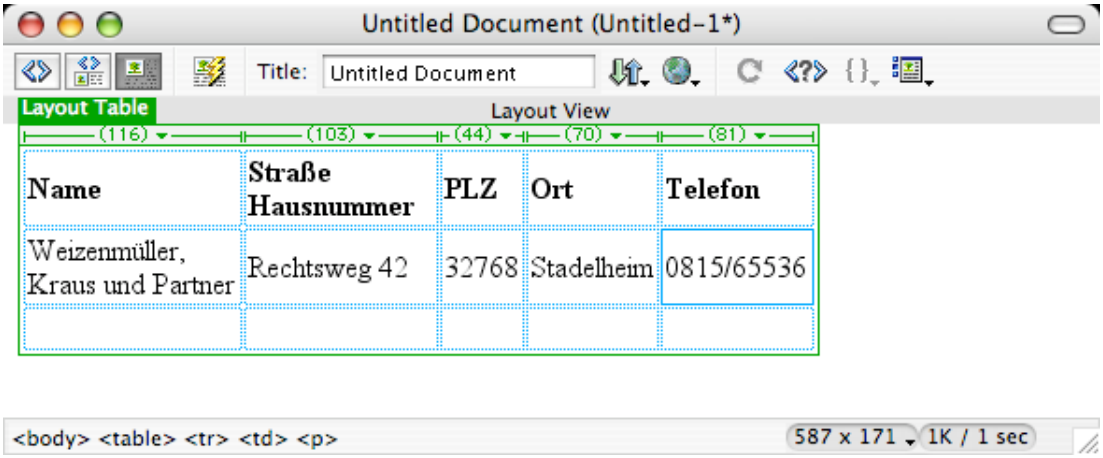

Abbildung 2.3: Ausfüllen einer Tabelle

Diese Art der Suche ist tendenziell eher langsam, da die Schlüsselwörter im regulären Ausdruck mit „oder“ verbunden sind. Gute Regexp-Engines können solche Ausdrücke zwar minimieren, aber nichtsdestoweniger dauert die Suche umso länger, je größer der Ausdruck wird. Wie oben gezeigt, müssen entweder Präzision oder Recall unter einem zu kurzen Suchausdruck leiden. Ist der Ausdruck zu generell, gibt es viele falsche Treffer, ist er zu genau, wird wenig gefunden. Sollen Präzision *und* Recall hoch sein, muß mit einem weitgehend vollständigen Lexikon gesucht werden.

2.2.3 Erkennen von Adreßstrukturen

Die Erkennung von tatsächlichen Adressen im Unterschied zu Segmenten mit Vielleicht-Adressen erfolgt bei Altfelds Programm in einem zweistufigen Prozeß. Zuerst wird die Plain-Text-Version der Webseite getaggt, das heißt es erfolgt ein Lexikon-Lookup der einzelnen Tokens (Wörter bzw. Wortgruppen), und wenn ein Token im Lexikon gefunden wird, wird es mit einem Tag annotiert, aus „Hans“ wird beispielsweise „<vn:Hans>“ [Altfeld, 36], womit das Wort als Vorname ausgewiesen wäre. Auch Altfeld sieht möglichst vollständige Lexika als notwendige Voraussetzung für gute Erkennungsergebnisse.

Nach dem Tagging wird mit einem regulären Ausdruck, der aus einer Grammatik generiert wird, nach kompletten Adressen im annotierten String gesucht und die Suchergebnisse als Adressen ausgegeben. Die externe, in einer eigenen Syntax erstellte Grammatik dient dazu, den daraus generierten regulären Ausdruck leichter pflegen zu können [Altfeld, 40]. Die Verwendung eines Compiler-Compilers bzw. Parser generators[3] lehnt Altfeld ab, „weil sie immer Quell-Code aus der Grammatik erstellen und damit eine Neu-Kompilierung erfordern“ [Altfeld, 24, 39]. (Siehe dazu Abschnitt 2.3.)

[3] Compiler-Compiler sind Programme, die eine Grammatik einlesen und daraus Maschinencode bzw. nochmals zu übersetzenden Quellcode einer Programmiersprache generieren.

2.3 Probleme und Verbesserungsvorschläge

Die grundsätzlichen Schwierigkeiten bei der automatischen Adreßerkennung und die Ausführungen zum Adreßaufbau sind in Altfelds Arbeit gut herausgearbeitet und decken sich großteils mit meinen eigenen Untersuchungen. Das erstellte Java-Programm wird von Altfeld selbst als „Prototyp“ bezeichnet, der leider ohne Eingriffe in den Quelltext auf aktuellen Systemen, die nicht Altfelds Entwicklungssystem entsprechen, nicht lauffähig ist. Die benutzten Lexika dienen als *proof of concept* und enthalten nur wenige Einträge, die zu dem beigefügten Korpus aus einem Dutzend Webseiten passen.

Altfeld macht selbst einige Vorschläge, wie die Adreßerkennung und -verarbeitung verbessert werden kann, auf die ich, zur eigenen Implementierung eines Adreßerkenners überleitend, eingehen möchte.

2.3.1 Ausgabeformat

Last things first: Altfeld schlägt vor, die Ausgabe der erkannten Adressen in XML einzubetten. „Dadurch entfällt die Konvertierung der ausgegebenen Adressen in das Datei-Format der jeweiligen Anwendung (z. B. Adress-Datenbank).“ [Altfeld, 61].

Wie oben bereits angeführt, ist XML eine Meta-Markupsprache. Es ist ein weitverbreiteter Irrtum, daß alleine die Verwendung von XML weitere Konvertierungen erübrige und jedes XML-fähige Programm die Daten sofort weiterverarbeiten könne. Die beiden Vorteile von XML sind, daß es einigermaßen menschenlesbar ist, solange die Strukturierung nicht zu kompliziert ist, und daß es mehr und mehr Programme gibt, die XML verarbeiten können, das heißt die einen *Parser* für XML mitbringen. Eine XML-fähige Datenbank oder ein sonstiges Programm, das die eingelieferten XML-Adreßdaten weiterverarbeiten soll, muß zusätzlich zu den Daten eine Beschreibung des XML-Formats erhalten (entweder in Schema oder DTD[4] oder auch in Relax NG[5]), und kann dann die Daten parsen und in den eigenen XML-Dialekt überführen. Wenn ein XML nutzendes Programm die Adreßdaten direkt weiterverarbeiten können soll, müssen sie im Dialekt des empfangenden Programms angeliefert werden. Mit dem Verpacken in XML-artige Syntax ist nichts gewonnen, im Gegenteil führt es im Vergleich zu anderen Arten der Strukturierung zu einem gewaltigen Aufblähen der Datenmenge durch die Verbosity[6] der Sprache.

Altfeld beschreibt das Ausgabeformat in einem Beispiel [Altfeld, 61]:

[4] Document Type Definition

[5] Relax NG ist die neueste, noch viel bessere XML-Definitionssprache, Genaueres unter <URL:http://www.relaxng.org/>

[6] „Wortfülle“, gemeint ist die Art des Taggings mit ausgeschriebenen End-Tags, die zur Menschenlesbarkeit des Formats beitragen soll.

```
<Adresse>
  <anrede>Herr</anrede>
  <vorname>Peter</vorname>
  <text-body>Mustermann</text-body>
  <strasse>Mustergasse</strasse>
  <haus-nr>1</haus-nr>
  <plz>12345</plz>
  <ort>Musterhausen</ort>
</Adresse>
```

Obiges Beispiel besteht aus 207 Zeichen. Dabei werden 159 Zeichen XML verwendet, um 48 Zeichen Adresse zu verpacken.

Wenn XML verwendet werden soll, um die erkannten Adressen auszugeben, würde es sich eher anbieten, einen XML-Dialekt für Adressen zu verwenden, zum Beispiel PATDL, die Postal Address Template Description Language, die von der UPU, der Universal Postal Union, in der 189 Länder vereinigt sind, entworfen wird (siehe [Lubenow]). Eine genaue Spezifikation des derzeitigen Standes des – entsprechend der Größe der entwerfenden Organisation – umfangreichen Standards findet sich unter [PATDL].

2.3.2 Grammatik

In [Altfeld, 60] wird darauf hingewiesen, daß Änderungen an den Meta-Anweisungen der verwendeten Grammatik zu Performance-Problemen führen können, und daß es je nach Justierung der Grammmatik zu Über- bzw. Untererkennung kommt. Eine mögliche Lösung dieses Problems wäre es, auf eine Grammmatik im eigentlichen Sinne zu verzichten, und Adressen zu „berechnen“. Dies erfordert, nach *Bestandteilen* von Adressen unabhängig voneinander zu suchen, und diese dann in einen örtlichen Bezug zu bringen durch Vergleich der Reihenfolgen und Nähe zueinander. Komplette Adressen könnten dann auch bei ungewöhnlicher Reihenfolge der einzelnen Bestandteile zusammengesetzt werden. Machbar ist das nur dann, wenn die Erkennung derselben nahezu fehlerfrei funktioniert. Wenn ein Bestandteil nicht gefunden wird, kann ein solches Vorgehen bei Adreßlisten zu falscher Zuordnung führen, was unbedingt zu vermeiden ist.

2.3.3 Implementierung

Altfeld begründet die Wahl der Programmiersprache Java mit dem „Aspekt der Wartbarkeit und Erweiterbarkeit“ [Altfeld, 85]. Dies stellt er dem Ergebnis einer Untersuchung gegenüber, die zu der Feststellung kommt, „daß ein Programmierer unabhängig von der verwendeten Programmmiersprache etwa immer die gleiche durchschnittliche Anzahl an Code-Zeilen pro Tag erstellt“ [a. a. O.]. Außerdem schlägt er die Verwendung eines Parser generators vor, um Grammatiken zu kompilieren, der aber zur Laufzeit Änderungen an der Grammatik ermöglichen sollte.

Die ideale Sprache hätte dementprechende folgende Eigenschaften:

- eine reiche Semantik
- Dynamizität
- erweiterbar in eine domänenspezifische Sprache (was externe Compiler-Compiler überflüssig macht)

All das bietet Common Lisp. Die Sprache ist seit 1994 ANSI-zertifiziert[7] und somit kein *moving target*, weshalb man nicht befürchten muß, in wenigen Jahren CL-Quellcode auf den neuesten Standard adaptieren zu müssen. Sie ist implementationsunabhängig, es existieren kommerzielle, der GPL[8] unterliegende und Public-Domain-Implementationen für nahezu jede Rechner- und Betriebssystemkombination. Der Standard umfaßt knapp 1000 Lexeme, kein Sprachstandard ist semantisch reichhaltiger. Lisp kann dynamisch kompiliert werden, das heißt Änderungen an einer Funktion erfordern nur Neukompilieren dieser Funktion, der Rest des laufenden Programms bleibt davon unberührt. Jede Funktionsdefinition wird Teil der Sprache, darüberhinaus ermöglichen Makros, die Sprache um neue Lexeme zu erweitern, die der Aufgabenstellung maßgeschneidert sind. Lisp ist ideal für das *rapid prototyping*, das schnelle Entwerfen und Testen von Lösungen, da der übliche Schreiben-Kompilieren-Testen-Ändern-Kompilieren-Testen-Zyklus statischer Sprachen entfällt und jederzeit Zugriff auf alle vom Programm verwendeten Datenstrukturen besteht.

Folgerichtig ist der Adreßerkenner dieser Arbeit in Common Lisp implementiert. Der Name steht für *Common-Lisp-Adreßerkenner* Der Funktionsumfang und die Performance von Clark wären mit einer anderen Sprache nicht im Rahmen einer Magisterarbeit erreichbar gewesen.

[7] CLOS, das Common Lisp Object System, ein Teil des Comomon-Lisp-Standards, war die erste ANSI-zertifizierte objektorientierte Sprache.

[8] GNU General Public License, siehe <URL:http://www.gnu.org/copyleft/gpl.html>

3 Clark – Implementierung eines Adreßerkenners

3.1 Ziele

3.1.1 Einsetzbarkeit für verschiedene Formate

Das Programm soll in der Lage sein, nicht nur Adressen in HTML-Seiten, sondern in jeder Art von Dateien zu erkennen, die in Plain-Text überführbar sind. In der gegenwärtigen Version ist es in der Lage, Text von jedem Programm zu übernehmen, das nach STDOUT[1] schreiben kann, und kann Plain-Text-Dokumente ohne Vorverarbeitung prozessieren.

Word-Dateien

Wie sich die Bürosoftwarelandschaft in den letzten Jahren darstellt, kann es als ziemlich sicher gelten, daß die meisten Texte und vor allem Briefe in dem einen oder anderen Microsoft-Word-Dateiformat vorliegen. Die Formate von „.doc“-Dateien unterscheiden sich je nach verwendeter Softwareversion deutlich. Neben dem Rich-Text-Format, das noch bis Word 6.0 eingesetzt wurde und heute plattformübergreifend von den meisten Textverarbeitungsprogrammen verstanden wird, gab es ab Word 7.0 verschiedene Binärformate, so daß auch ältere Versionen desselben Programms Dateien, die im Standardformat einer aktuelleren Version erstellt wurden, nicht in jedem Fall lesen können. Die aktuelle Version für Windows, Word 2003, erstellt wahlweise Dateien in einem XML-Dialekt, was sie im Prinzip leichter einer Konvertierung zugänglich macht.

Das für Clark am besten geeignete Tool, um schnell ein passendes Format zu erzeugen, ist das Kommandozeilenprogramm `antiword` [antiword]. Es erzeugt Plain-Text-Output[2] aus Word-Dateien von Version 2 bis 2002. Mit bestimmte Formatierungen kann es allerdings Probleme geben. In solchen Fällen empfiehlt es sich, direkt aus Word ins Plain-Text-Format zu exportieren. Auf den Plattformen, die von Microsoft unterstützt werden, gibt es dafür jeweils Möglichkeiten, dies automatisiert für eine Menge von Dateien zu erledigen. Unter Apple-Betriebssytemen ist dies mit AppleScript machbar, Word-Versionen

[1] STDOUT ist der Standard-Ausgabekanal auf Unix-Systemen. Die meisten Kommandozeilentools können nach STDOUT ausgeben.

[2] alternativ auch Postscript und Docbook

für Windows bieten Visual Basic als Skriptsprache. Clark würde in diesem Fall die erzeugten Plain-Text-Dateien direkt verarbeiten. Diese Art der Umwandlung ist allerdings wesentlich aufwendiger und langsamer, als aus Clark heraus ein Konvertierungstool wie `antiword` aufzurufen.

In Fällen, in denen weder eine Batch-Konvertierung noch der Einsatz von `antiword` möglich ist, kann man auf das Kommandozeilenprogramm `strings` ausweichen, das zum Standardumfang der meisten Unixartigen gehört. `strings` gibt ohne jegliche Formatierung sämtliche in einer Datei enthaltenen Strings (Portionen von Text) aus, unabhängig davon ob sie tatsächlich zur gewünschten Ausgabe des Dateiformats gehören oder Kommentare oder sonstige Metainformationen sind. Dementsprechend ist es deutlich schwieriger, mit dem Output von `strings` etwas Vernünftiges anzufangen, denn `strings` weiß auch nicht, wie die Strings codiert sind.

PDF-Dateien

PDF, das Portable Document Format von Adobe Systems Inc., ist ein offenes Dateiformat, das auf nahezu jeder Plattform gelesen werden kann. Neben Adobes Reader (früher: Acrobat Reader) gibt es eine Menge kostenloser und kommerzieller Programme, um PDF zu erzeugen und zu betrachten.

Für PDF existiert ein ähnliches Programm wie `antiword`: `pdftotext` [pdftotext]. Es kann sowohl unkomprimierte und komprimierte als auch paßwortgeschützte PDF-Dateien konvertieren (das Paßwort muß natürlich vorhanden sein). `pdftotext` ist Teil der Xpdf-Programmsammlung (Xpdf ist ein freier PDF-Betrachter für X-Window).

Es ist nicht möglich, mit `pdftotext` kopiergeschützte Dateien zu konvertieren. Ein Versuch ergibt nur eine entsprechende Fehlermeldung. PDF-Dateien können mit Meta-Informationen wie Autor, Titel, erzeugendes Programm versehen sein. `pdftotext` gibt diese Informationen standardmäßig zwar nicht aus, aber mit der Option `-htmlmeta` konvertiert es die Datei in HTML, wobei im Head-Bereich die Meta-Informationen als Meta-Tags angegeben sind, und der eigentliche Inhalt des PDF im Body landet, in `PRE`[3]-Tags eingebettet.

Es existieren auch kommerzielle Konvertierungsprogramme, aber die konnten mangels vorhandener Lizenzen nicht im Rahmen dieser Arbeit getestet werden.

Plain Text

Reiner Text ist das Format, das Clark intern zur Verarbeitung benutzt und damit voll unterstützt. Bei den oben angegebenen Konvertern können Umwandlungsverluste auftreten: verlorene Formatierungen ebenso wie verschwundene oder kaputte Umlaute. Das einzige Problem bei im Nur-Text-Format vorliegenden Dateien kann das Encoding sein.

[3] preformatted, also Nur-Text, der, wie er ist, wiedergegeben wird

Die verwendeten Lexika liegen im ISO-8859-1-Format[4] vor; haben die zu verarbeitenden Dateien ein anderes Encoding, müssen sie vorher umcodiert werden. Schnell arbeitende Kommandozeilentools wie `iconv` oder `recode` stehen dafür zur Verfügung.

Webseiten

Die Abbildungen 3.1 und 3.2 zeigen Screenshots zweier Versionen der gleichen Tabelle, die einmal als ungerahmt und einmal als gerahmt (`table border="1"`) definiert wurde. Der Quelltext der Webseite liegt der Arbeit bei (tabelle.html). Die beiden Tabellen dienen im Folgenden als Beispiel für breite Tabellen.

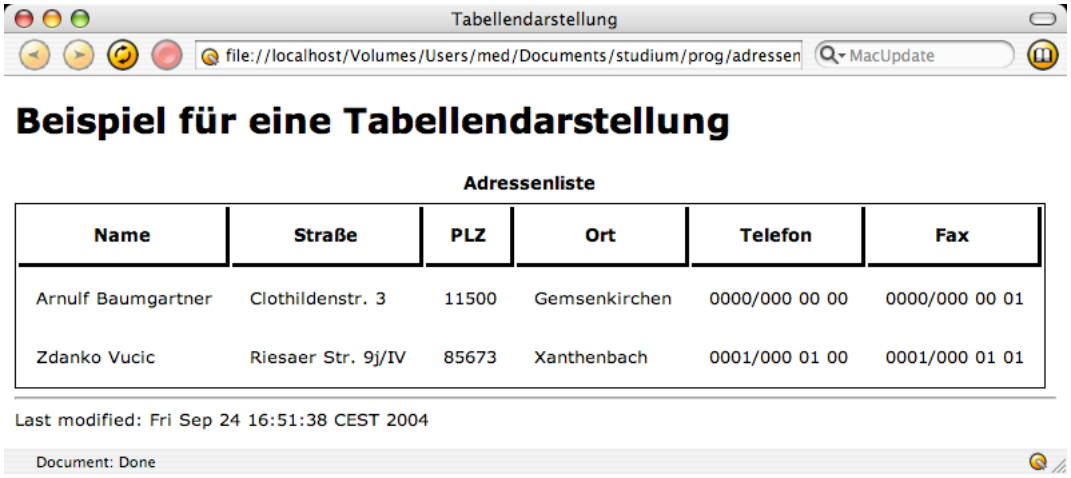

Abbildung 3.1: Webseite mit langer Tabelle

Webseiten müssen ebenso wie PDF- oder Word-Dateien erst in reinen Text umgewandelt werden, bevor die Adreßerkennung beginnen kann. Die Konverter für HTML sind deutlich ausgereifter, fehlerfreier und weiter konfigurierbar als die für die bisher angesprochenen Dateiformate. Das liegt daran, daß die Konvertierung ein Nebeneffekt einer anderen Aufgabe ist: der Darstellung von Webseiten auf Textkonsolen bzw. ihrer Aufbereitung für Screenreader oder andere textbasierte Ausgabegeräte. Zwei verbreitete Textbrowser verfügen über besonders ausgefeilte Konvertierungsfunktionen: `Lynx` und `w3m` (siehe [Lynx] bzw. [w3m]). Wenn sie mit der Option `-dump` aufgerufen werden, starten sie nicht im Browsermodus, in dem ähnlich navigiert werden kann wie mit einem grafischen Browser, sondern geben den Text der als Argument übergebenen Seite nach STDOUT aus. `w3m` versucht dabei, Tabellen (auch Layouttabellen!) so vollständig wie möglich ins Textformat zu übersetzen. Ungerahmte Tabellen, die nicht breiter als 80

[4] Auch bekannt als Latin-1. Verwandt mit Windows-1252 und Latin-9 bzw. ISO-8859-15. All diese Encodings sollten ohne Umwandlung prozessiert werden können, im Zweifelsfall ist aber immer erst zu konvertieren.

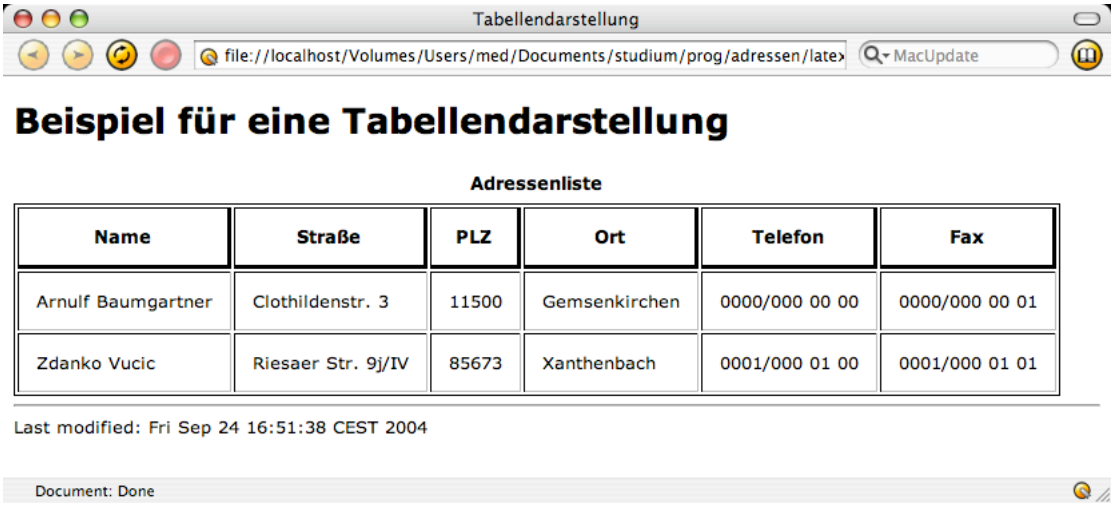

Abbildung 3.2: Webseite mit langer, gerahmter Tabelle

Zeichen werden[5], sehen dabei bei beiden sehr ähnlich aus (Abbildungen 3.3 und 3.4), wobei `Lynx` noch die Tabellenbenennung mit angibt.

```
                  Beispiel für eine Tabellendarstellung

   CAPTION: Adressenliste

          Name              Straße         PLZ       Ort
   Arnulf Baumgartner Clothildenstr. 3   11500 Gemsenkirchen
   Zdanko Vucic       Riesaer Str. 9j/IV 85673 Xanthenbach
     ------------------------------------------------------------------

   Last modified: Fri Sep 24 16:51:38 CEST 2004
```

Abbildung 3.3: Lynx-Ausgabe einer Webseite mit kurzer Tabelle

`Lynx` stellt gerahmte Tabellen genauso dar wie ungerahmte. `w3m` dagegen zeichnet einen ASCII-Rahmen (Abbildung 3.5).

Wird die Tabelle breiter als die vorgegebene Textbreite, wird sie umbrochen, und dabei verhalten sich die beiden Programme deutlich anders. Lynx umbricht zeilenweise (Abbildung 3.6), während w3m *zellenweise* umbricht (Abbildungen 3.7 und 3.8).

Dies macht w3m unbrauchbar als Konvertierungstool, wenn der Output programmatisch weiterverwendet werden soll, da hier tatsächlich Daten durchmischt werden. Die erste Tabellenzeile aus Abbildung 3.8 ergibt diesen Datenstrom für die erste Adresse:

[5]Die meisten Textkonsolen sind auf 80 Zeichen Breite voreingestellt. Die Darstellungsbreite kann bei `Lynx` mit der Option `-width=`*number* angepaßt werden; für w3m lautet die Option `-cols` *number*

```
Beispiel für eine Tabellendarstellung

                     Adressenliste
       Name              Straße        PLZ       Ort
Arnulf Baumgartner  Clothildenstr. 3   11500 Gemsenkirchen
Zdanko Vucic        Riesaer Str. 9j/IV 85673 Xanthenbach

--------------------------------------------------------------------------------
Last modified: Fri Sep 24 16:51:38 CEST 2004
```

Abbildung 3.4: w3m-Ausgabe einer Webseite mit kurzer Tabelle

```
Beispiel für eine Tabellendarstellung

                      Adressenliste
+-----------------------------------------------------------+
|       Name       |      Straße       | PLZ |     Ort     |
|------------------+-------------------+-----+-------------|
|Arnulf Baumgartner|Clothildenstr. 3   |11500|Gemsenkirchen|
|------------------+-------------------+-----+-------------|
|Zdanko Vucic      |Riesaer Str. 9j/IV |85673|Xanthenbach  |
+-----------------------------------------------------------+
--------------------------------------------------------------------------------
Last modified: Fri Sep 24 16:51:38 CEST 2004
```

Abbildung 3.5: w3m-Ausgabe einer Webseite mit kurzer, gerahmter Tabelle

```
                 Beispiel für eine Tabellendarstellung

CAPTION: Adressenliste

       Name              Straße        PLZ       Ort         Telefon     Fax
Arnulf Baumgartner Clothildenstr. 3   11500 Gemsenkirchen 0000/000 00 00
0000/000 00 01
Zdanko Vucic       Riesaer Str. 9j/IV 85673 Xanthenbach   0001/000 01 00
0001/000 01 01
  ------------------------------------------------------------------

Last modified: Fri Sep 24 16:51:38 CEST 2004
```

Abbildung 3.6: Lynx-Ausgabe einer Webseite mit langer, gerahmter Tabelle

```
Beispiel für eine Tabellendarstellung

                                Adressenliste
      Name            Straße       PLZ       Ort         Telefon         Fax
Arnulf           Clothildenstr. 3 11500 Gemsenkirchen 0000/000 00   0000/000 00
Baumgartner                                          00             01
Zdanko Vucic     Riesaer Str. 9j/ 85673 Xanthenbach   0001/000 01   0001/000 01
                 IV                                  00             01

--------------------------------------------------------------------------------
Last modified: Fri Sep 24 16:51:38 CEST 2004
```

Abbildung 3.7: w3m-Ausgabe einer Webseite mit langer Tabelle

```
Beispiel für eine Tabellendarstellung

                                Adressenliste
+--------------------------------------------------------------------------------+
|     Name      |    Straße     | PLZ |     Ort     |  Telefon   |    Fax     |
|---------------+---------------+-----+-------------+------------+------------|
|Arnulf         |Clothildenstr. |11500|Gemsenkirchen|0000/000 00 |0000/000 00 |
|Baumgartner    |3              |     |             |00          |01          |
|---------------+---------------+-----+-------------+------------+------------|
|Zdanko Vucic   |Riesaer Str. 9j|85673|Xanthenbach  |0001/000 01 |0001/000 01 |
|               |/IV            |     |             |00          |01          |
+--------------------------------------------------------------------------------+
--------------------------------------------------------------------------------
Last modified: Fri Sep 24 16:51:38 CEST 2004
```

Abbildung 3.8: w3m-Ausgabe einer Webseite mit langer, gerahmter Tabelle

```
| Arnulf | Clothildenstr. | 11500 | Gemsenkirchen | 0000/000 00 |
  0000/000 00 | | Baumgartner | 3 | | | 00 |01 |
```

Selbst wenn man die ASCII-Rahmen entfernt, ist keine Zuordnung der einzelnen Zellenbestandteile mehr möglich, außer mit großem Rechenaufwand. `Lynx` tut „das Richtige“, wenn es um automatische Weiterverwertung der Daten geht.

3.1.2 Automatisches Prozessieren einer großen Menge von Dateien

Clark ist darauf ausgelegt, lange Zeit zu laufen und eine große Anzahl von Dateien unbeaufsichtigt zu bearbeiten. Alle nötigen Datenstrukturen werden im Arbeitsspeicher vorgehalten; beliebige neue Input- und Outputmethoden zu schreiben ist einfach. Durch die interaktive Entwicklungs- und Laufzeitumgebung kann der Prozeß jederzeit überwacht werden.

3.1.3 Möglichst hohe Präzision

Das Entwicklungsziel war, eine hohe Präzision zu erreichen, was sich unter Umständen negativ auf den Recall auswirken kann. Clark soll keine falschen Adressen erkennen. Jeder Adreßbestandteil hat einen eigenen, optimierten Matcher, es wird nur in Ausnahmefällen mit regulären Ausdrücken gearbeitet. Einige Matcher sind von anderen abhängig – so wird die Namenserkennung nur gestartet, wenn bereits andere Adreßbestandteile erkannt sind, und sie wird nur auf das direkte Umfeld der anderen Teile angewendet.

3.1.4 Weitgehende Modularität und Anpaßbarkeit auf verschiedene Bedürfnisse

Das Programm ist modular aufgebaut. Neue Matcher können jederzeit hinzugefügt, aus den vorhandenen kann bedarfsgerecht ausgewählt werden. Es kann als Library von anderen Lisp-Programmen verwendet werden, ist interaktiv über den Lisp-REPL[6] bedienbar, und es kann über ein Webinterface bedient werden. Bei häufiger Wiederverwendung bestimmter Input-Output-Methoden wäre es problemlos möglich, eine grafische Bedienoberfläche hinzuzufügen. Der Einsatz einer kommerziellen Common-Lisp-Implementation wie Lispworks[7] oder Allegro Common Lisp[8] würden es erlauben, das Programm als Standalone-Executable für Linux, UNIX, Mac OS X und Windows auszuliefern. Eine Weiterentwicklung in diverse Richtungen ist möglich.

[6] Read-Eval-Print-Loop, die „Shell“ von Lisp
[7] <URL:http://www.lispworks.com/>
[8] <URL:http://www.franz.com/>

3.1.5 Interaktive (Weiter-) Entwicklung und Nutzung

Eine der Stärken von Common Lisp liegt in der Interaktivität. Man kann aus Common-Lisp-Programmen zwar auch ausführbare Dateien mit Kommandozeilen- oder grafischem Interface machen, beraubt sich damit aber eines großen Vorteils gegenüber herkömmlicher Programmbenutzung.

Abbildung 3.9 zeigt einen Screenshot einer typischen Clark-Session. Das Programm läuft dabei auf einem entfernten Rechner. Die Verbindung wurde über `ssh`[9] hergestellt. Ab einem gewissen Zeitpunkt in der Entwicklung des Programms bot es sich an, Entwicklungs- und Laufzeitort zu trennen. Clark braucht zur Zeit etwa 500 MB RAM für sich alleine, dazu war eine schnelle Internetanbindung für die Web-Tests nötig, und das Webinterface sollte ständig erreichbar sein. Die Entwicklung selbst fand auf einem Notebook stand.

Auf dem Remote-Rechner wird eine `screen`-Sitzung gestartet [screen]. Diese ermöglicht es, die Verbindung zum entfernten Rechner zu trennen und später wieder aufzubauen, ohne daß die in `screen` laufenden Prozesse beendet werden. Wenn man sich wieder mit `ssh` verbindet und `screen -r` aufruft, kann man an der Stelle weitermachen, an der man beim letzen Mal aufgehört hatte.

Hinweise zur Notation von Emacs-Kommandos

Der Emacs kann vollständig über die Tastatur bedient werden. Da viele Kommandos aus mehreren Tastenkürzeln in Verbindung mit verschiedenen Modifikator-Tasten bestehen, hat sich dafür eine Kurzschreibweise etabliert.

`C-x` bezeichnet die Tastenkombination Ctrl - x

`M-x` heißt Meta - x [a]

`RET` steht für ↩ .

Beispiel: `C-c RET x`: zuerst Ctrl - c , dann ↩ , dann x .

[a] Auf PC-Keyboards liegt die Meta -Funktion meist auf Alt ; auf Apple-Rechnern auf Command (der „Apfel“-Taste). Ist keine solche Taste vorhanden, oder ist sie anderweitig belegt, kann Meta - x durch Esc und danach x emuliert werden.

In `screen` startet man einen Emacs im Terminal-Modus und wechselt ins Clark-Quellcodeverzeichnis. `M-x slime` startet die Common-Lisp-IDE[10] SLIME [SLIME] mit dem voreingestellten Lisp. Im Fall von Clark ist das CMUCL[11] [CMUCL]. Clark verfügt über einen einfachen Lademechanismus mittels ASDF[12] [ASDF] und kann mit nur einem Befehl geladen werden, wobei Teile, die sich seit dem letzten Mal geändert haben, automatisch neu kompiliert werden.

[9] Secure Shell, siehe <URL:http://www.openssh.org/>
[10] Integrated Development Environment – Entwicklungsumgebung
[11] Carnegie-Mellon University Common Lisp
[12] Another System Definition Facility

Abbildung 3.9: Remote-Session: Clark/CMUCL/SLIME/Emacs/screen

Es ist zwar auch möglich, CMUCL direkt von der Kommandozeile (auch in einer `screen`-Session) zu starten und Clark mit dem gleichen Befehl dort zu laden, aber gerade die Kombination mit einem selbst in Lisp programmierbaren Editor wie Emacs macht die Entwicklung zu einem sehr effizienten Prozeß. Abbildung 3.9 zeigt ein Beispiel dafür. Im unteren Buffer befindet sich der Lisp-REPL, im oberen ist eine Clark-Datei geöffnet. Wenn man nun an einer der Funktionen eine Änderung vornimmt, ist es nicht einmal nötig, die Datei erst zu speichern. Mit `C-c C-c` wird die Funktion dem Common-Lisp-Prozeß übergeben und kompiliert. Sie steht sofort zur Verfügung und wird auch von bereits laufenden Prozessen übernommen. SLIME zeigt zu allen Funktionen die nötigen Argumente während der Eingabe im Emacs-Minibuffer an – auch zu Funktionen, die selbst geschrieben wurden. Mit `C-c C-d h` wird im Webbrowser die Dokumentation der aktuellen Funktion im Common-Lisp-Hyperspec aufgerufen. SLIME stellt aufbereitete Schnittstellen zu den leistungsfähigen Debuggern der Common-Lisp-Implementierungen zur Verfügung. Gerade bei der Programmierung der Matcher hat sich diese Entwicklungsumgebung als hilfreich erwiesen. Werden Fehler an einer Stelle gefunden, können sie sofort behoben und das Testen fortgesetzt werden.

3.2 Umsetzung mit Fokus auf Erkennung in Webseiten

Um die gewünschte hohe Erkennungspräzision zu erreichen, war es nötig, umfangreiches Datenmaterial zu erstellen. Einiges davon war frei im Web erhältlich, anderes Rohmaterial lag bereits am CIS vor. Allen Rohdaten war gemeinsam, daß sie im vorgefundenen Zustand nicht sinnvoll nutzbar waren.

3.2.1 Listen

Alle vom Programm benutzten Daten befinden sich um Unterverzeichnis „lists“.

Postleitzahl-Ort-Kombinationen

plz-ort.txt Die momentan vom Programm genutzte Liste der Postleitzahl-Ort-Kombinationen hat 17.556 Einträge. Dabei war zu beachten, daß Orte mit und ohne und auch mit abgekürzten Zusätzen vorkommen können. Alle Orte wurden deshalb in ein einheitliches Format gebracht (ausgeschriebene Zusätze), dann wurden manuell die Varianten hinzugefügt. Beispiele:

```
01776 Hermsdorf/Erzgeb.
01776 Hermsdorf/Erzgebirge
02748 Bernstadt a. d. Eigen
02748 Bernstadt auf dem Eigen
```

plz-ort-mit-entitaeten.txt Diese Liste wurde aus *plz-ort.txt* generiert. Sie enthält zusätzlich die Ortsnamen mit durch HTML-Entitäten ersetzten Umlauten. Die Funktion `read-convert-count-write` (Argumente: *Inputdatei Outputdatei*) nimmt die Konvertierung vor, zählt die Einträge und gibt die Anzahl sowie die Länge des längsten Ortsnamens aus.

Straßennamen

strassen-bereinigt.txt Diese Liste enthält 212.054 rechtschreiblich korrekte Straßennamen. Sie wurde Eintrag für Eintrag auf Korrektheit überprüft. Die Namen sind dabei ausgeschrieben, außer bei der Endung „Straße“, die in „Str“ bzw. „str“ (ohne Punkt) abgekürzt ist.

strassen-ausgeschrieben.txt Die Liste wurde aus *strassen-bereinigt.txt* generiert. Sie enthält zusätzlich die Namen mit „str“-Endung in expandierten Varianten: `str` wird zu `str.` und `straße`. Die Funktion `schreib-strasse-aus` (Argumente: *Inputdatei Outputdatei*) dient zur Konvertierung.

strassen-verheert.txt, strassen-mappings.lisp Erstere Datei enthält die Straßennamen aus *strassen-bereinigt.txt* und zusätzlich Varianten mit üblichen Rechtschreibfehlern. Außerdem enthält sie generierte Straßennamen. Während der Tests stellte sich heraus, daß einige Straßen nicht im Datenbestand vorhanden waren, obwohl es Versionen mit anderem Straßengrundwort gab – so gab es den „Lil-Dagover-Weg“, aber nicht den „Lil-Dagover-Ring“. Als Konsequenz werden solche Straßennamen nun von der Funktion `verheere-strassen` erzeugt. Neben *strassen-verheert.txt* entsteht dabei noch die Datei *strassen-mappings.lisp* in einem menschenlesbaren Format, das vom Lisp-Reader direkt eingelesen werden kann. Diese Datei muß anschließend durch `kategorisiere-verheerte-strassen` bereinigt werden. Die Funktion vergleicht die Einträge mit der originalen Straßenliste. In *strassen-mappings.lisp* ist eingetragen, ob es sich bei einem Straßennamen um einen originalen oder einen generierten handelt, und ob die Rechtschreibung korrekt ist. Diese Informationen werden zur Zeit vom Programm noch nicht verwertet, können aber als Grundlage für eine künftige Rechtschreibprüfung dienen.

3.2.2 Inputvorverarbeitung

```
(defmacro with-lynx-web-dump ((string URL) &body body)
  "Sends 'filename' through lynx -dump and creates a
string stream from lynx's output."
  `(let ((process (gensym))
         (stream (gensym))
         (,string (make-array 5000 :element-type 'base-char
                                   :fill-pointer 0 :adjustable t)))
```

```
(setf process
      (ext:run-program *lynx*
                       (append *lynx-options* (list ,URL))
                       :output :stream :error nil :wait nil))
(setf stream (ext:process-output process))
(with-output-to-string (s ,string)
  (do ((c (read-char stream nil 'eof) (read-char stream nil 'eof
      )))
      ((eq c 'eof))
    (if (eql c #\Newline)
        (format ,string "~A~A" #\Newline #\Newline)
        (format ,string "~A" c))))
(ext:process-close process)
,@body))
```

Listing 3.1: Beispiel für ein Inputmakro

Listing[13] 3.1 zeigt ein Inputmakro für Input von HTML-Dateien. `with-lynx-web-dump` nimmt den Namen einer Variablen und einen URL (String) als Argumente. Der URL kann auch auf die lokale Platte verweisen (`file://`). Der Output von `Lynx` wird in einen String-Stream geschrieben, der anschließend über den als Argument 1 vergebenen Namen weiterverarbeitet werden kann.

Der String-Stream wird der Funktion `tokenize-lynx-dump` übergeben. Diese erzeugt daraus eine Liste von Tokens, eine Liste von Startpositionen dieser Tokens im Stream und einen Integer-Wert, der die Anzahl von Tokens enthält. Diese Daten werden von den Matchern verarbeitet.

3.2.3 Datenstrukturen

In Clark werden aus Performancegründen Hashtabellen für das Token-Matching verwendet. Diese sind wie alle globalen Variablen in *globals.lisp* definiert. Postleitzahl und Ort stehen in `*orthash*`, Schlüssel sind die Postleitzahlen, Werte eine Liste der möglichen Orte zur Postleitzahl. Die Straßen sind auf sechs Hashes verteilt – der längste Straßenname besteht aus sechs Einzelwörtern. In `*streethash-1*` sind die Schlüssel die ersten Wörter, als Wert kann eines der Keywords `:end`, `:more` oder `:both` stehen, welches angibt, ob der Straßenname mit diesem Wort zu Ende ist, ob noch ein Wort folgen muß oder kann.

Von den Matchern gefundene Adreßbestandteile werden als Liste in einem Vektor mit Fill-Pointer abgelegt. Die Listen sind wie folgt aufgebaut:

Position Position des ersten Characters des Fundstück im Original-Stream

Länge Länge des Fundstücks

[13] Alle in den Listings blau gedruckten Funktionen gehören nicht zum Sprachumfang von Common Lisp, sondern sind Clark-Funktionen.

Typ Typ des Fundstücks (ein Keyword wie `:strasse`, `:ort` oder `:fax`)

Token-Nummer Nummer des Tokens (durchgezählt von 1 bis Länge der Token-Liste)

Fundstück Der Rest ist das eigentliche Fundstück und kann beliebig viele Elemente beliebiger Art aufnehmen.

Nachdem alle Matcher durchgelaufen sind, können die Daten im Match-Vektor sortiert und beliebig weiterverarbeitet werden. Auch die Matcher selbst können den Match-Vektor auslesen und bearbeiten, etwa um Duplikate zu entfernen, Fundstücke zusammenzufassen oder um nur Fundstücke im Bereich vorher gefundener anderer Adreßbestandteile zu speichern und andere zu verwerfen.

Zur Zeit landen die endgültigen Suchergebnisse in einem weiteren Vektor, dem Ergebnisvektor, der als Elemente Strukturen enthält. Zur Definition der Ergebnisstruktur siehe Listing 3.2.

```
(defstruct address
  (name :missing)
  (strasse :missing)
  (ort :missing)
  (tel :missing)
  (fax :missing)
  (email :missing))
```

Listing 3.2: Definition der Ergebnisstrukturen

Listing 3.3 zeigt eine API[14]-Funktion, die alle Adreßmatcher testet (bis auf die, die nach URLs über Name oder Anchortext suchen).

```
(defun test-assembly-web (URL &key nocleanp)
  (with-match-and-result-vectors (vec result)
    (with-lynx-web-dump (in URL)
      (multiple-value-bind (tok pos len) (tokenize-lynx-dump in)
        (find-plz-ort tok pos len vec)
        (find-streets tok pos len vec :nocleanp nocleanp)
        (find-postfach tok pos vec)
        (find-email tok pos len vec)
        (find-telcom tok pos len vec)
        (find-name tok pos len vec :nocleanp nocleanp)))
    (cleanup-match-vector vec)
    (assemble-address vec result)
      result))
```

Listing 3.3: Testfunktion für alle Adreßmatcher

[14] Application Programming Interface – der Teil des Programms, der von außen angesprochen werden soll und kann.

`with-match-and-result-vectors` legt den Match- und den Ergebnisvektor an, `tokenize-lynx-dump` erzeugt aus dem von `with-lynx-web-dump` aufgebauten String die Inputdaten für die Matcher. Diese tun nacheinander ihre Arbeit. Am Ende wird der Match-Vektor aufgeräumt und aus den dort vorhandenen Daten der Ergebnisvektor mit den Adreßstrukturen erzeugt. Im Beispiel werden diese einfach nach STDOUT ausgegeben; genausogut könnte man sie aber in eine Datei schreiben oder als Ergebnis ans Webinterface liefern. Solange man sich an die von den einzelnen Funktionen gewünschten Datentypen hält, kann man jedes einzelne Teil durch ein anderes ersetzen.

3.2.4 Matching

Jeder Matcher verwendet einen eigenen effizienten Algorithmus zur Erkennung. Die meisten können autark verwendet werden, bei einigen müssen vorher andere am Werk gewesen sein. Die Namenserkennung läuft nur, wenn zuvor andere Adreßbestandteile gefunden wurden, weil sie deren Art und Position berücksichtigt.

Ortserkennung

`find-plz-ort` erkennt Postleitzahl-Ort-Kombinationen. Es sucht erst nach Ziffern. Wenn ein Token eine Ziffer enthält, wird die Tokenlänge geprüft. Ist es lang genug, wird die Zahl ermittelt und mit `*orthash*` verglichen. Folgt darauf einer der passenden Werte aus dem Hash, gilt der Ort als gefunden und wird im Match-Vektor abgelegt.

Straßenerkennung

`find-streets` erkennt Straßennamen und Hausnummern. Jedes Token wird mit den Schlüsseln in `*streethash-1*` verglichen. Gibt es einen Treffer, wird je nachdem, ob es ein `:end`-Token ist oder nicht, das nächste Token mit `*streethash-2*` verglichen usw. Liegt ein `:end`-Token vor, wird geprüft, ob der ganze gefundene Straßenname in `*maphash*` vorhanden ist. Entspricht dann das nächste Token noch dem regulären Ausdruck für Hausnummern, gilt eine Straßenangabe als gefunden und wird im Match-Vektor abgelegt. An dem Prozeß sind die rekursive Funktion `compare-tokens-with-streethashes` und die Ausputzer-Funktion `clean-streets` beteiligt, wobei letztere zu Testzwecken mit einem Flag[15] deaktiviert werden kann.

Postfacherkennung

`find-postfach` erkennt Postfachnummern, die intern als Typ `:strasse` abgelegt werden. Es sucht nach Postfachbezeichnern und sammelt danach alle folgenden Ziffern ein. Die mit einem Flag deaktivierbare Ausputzerroutine entfernt Ziffern, die die Nummer

[15] genauer: ein Keyword-Argument

eine bestimmte Länge überschreiten lassen. Das ist nötig, um Überschneidungen mit Postleitzahlen zu korrigieren.

Telefon- und Faxnummernerkennung

`find-telcom` erkennt Telefon- und Faxnummern. Es sucht nach entsprechenden Bezeichnern und sammelt danach alle folgenden Ziffern und Zeichen, bis ein Buchstabe gefunden wird. Somit werden auch Trennzeichen wie „/“ oder Klammern erfaßt.

E-Mail-Adreßerkennung

`find-email` erkennt E-Mail-Adressen. Dabei macht es sich eine Besonderheit der `Lynx`-Dumps zunutze: `Lynx` gibt nach der textuellen Repräsentation der Webseite noch eine numerierte Liste von Referenzen aus, die alle klickbaren Links und E-Mail-Adressen der Seite enthält. Jede E-Mail-Adresse erhält das Prefix „mailto:“. Nach diesem wird gesucht, die folgende E-Mail-Adresse eingesammelt und ihre Position anhand der Referenznummer ermittelt, so daß sie der Anschrift, falls vorhanden, zugewiesen werden kann.

Namenserkennung

Obwohl die Namenserkennung nur rudimentär implementiert ist, nimmt sie so viele Codezeilen ein, daß sie nicht mehr in *matching.lisp*, sondern in eine eigene Datei, *matching-names.lisp* ausgelagert wurde.

Die Funktion `find-names` versucht, Namen zu erkennen. Sie ruft wiederum drei Unterfunktionen auf, die Namensbestandteile erkennen, die am Anfang eines Firmennamens stehen können („Firma“, „Bürogemeinschaft“), in der Mitte („Inhaber“, „Abteilung“) oder am Ende („GmbH“, „GbR“). Um je nach Funktion den Anfang oder das Ende des Namens oder beides zu erkennen, greifen die Funktionen auf vorhandene Informationen über andere, gefundene Adreßbestandteile zurück oder sammeln alles ein, bis ein Abstand zum nächsten Token gefunden wird, der größer ist als ein (Leer-)zeichen. Um diesen Abstand auch bei Zeilenwechseln sicherzustellen, werden diese bereits beim Prozessieren des `Lynx`-Outputs verdoppelt. Wird ein Name erfolgreich erkannt, kann dies für die momentane Version von Clark als „Zuckerl“ gelten. Spätere Versionen sollten eine bessere Namenserkennung erhalten, aber dafür ist mehr Forschung nötig, als in eine Magisterarbeit paßt.

Verweiserkennung

Für die Bearbeitung von Webseiten sind noch zwei zusätzliche Matcher implementiert: `find-address-url` und `find-address-anchor-text`. Alle Matcher haben einen Integer

als Rückgabewert. Ist dieser 0, wurde nichts gefunden, wenn größer als 0, gab es Treffer. Dies kann für bedingten Einsatz von Matchern verwendet werden. Wird auf einer Webseite keine Adresse gefunden, können die Verweis-Matcher aufgerufen werden, die anhand von typischen Wörtern in der Linkbeschreibung bzw. im Namen des URL Links auf Seiten erkennen können, die möglicherweise eine Adresse enthalten. Clark folgt in der momentanen Version solchen Links nicht (er enthält keinen Crawler), kann sie aber ausgeben. Im Webinterface kann man auf solche Angaben klicken und eine neue Suche auf der verlinkten Seite auslösen.

3.3 Installation und Benutzung

Nach etwa einem Drittel der Entwicklungsphase wurde ein Subversion-Repository[16] für das Programm angelegt. Dieses ermöglicht Versionierung und Zugriffe von verschiedenen Orten und Rechnern aus, wobei die Codebasis konsistent gehalten wird. Folgende Erklärungen beziehen sich auf Version 84.

Clark wurde unter Mac OS X „Panther“ und verschiedenen Linux-Distributionen mit Kernel 2.4 und 2.6 entwickelt und getestet. Als Common-Lisp-Implementation kam CMUCL in Versionen 18e und 19a zum Einsatz, auf das die Wahl fiel wegen der Verfügbarkeit für die meisten Unix-Systeme, der freien Lizenz (Public Domain), und des guten Compilers („Python“, der Name wurde lang vor der Entwicklung der gleichnamigen Programmiersprache gewählt: CMUCL wird seit Anfang der 80er Jahre entwickelt), der effizienten Maschinencode erzeugt. Insbesondere verfügt CMUCL über sehr schnelle Hash-Tabellen, nützliche Erweiterungen zur Kommunikation mit dem Betriebssystem, und kann auch die großen Datenstrukturen in Clark verwalten. Eine Beschränkung in der Beta-Version für OS X[17] ermöglicht zur Zeit nur eine eingeschränkte Nutzung des Adreßerkenners, so daß es empfehlenswert ist, das Programm unter Linux oder FreeBSD zu installieren.

Der Arbeit liegt eine CD bei, die den Quellcode von Clark enthält, CMUCL in Versionen für Mac OS X 10.3 und Linux/X86, und die nötigen Zusatzbibliotheken für den Adreßerkenner und das Webinterface. Besonders zu erwähnen sind dabei *cl-ppcre*, *tbnl* und *cl-who*, alle drei geschrieben von Dr. Edmund Weitz [Weitz].

CL-PPCRE *portable Perl-compatible regular expressions for Common Lisp.* Die in reinem ANSI-CL geschriebene Bibliothek implementiert reguläre Ausdrücke in Perl-Syntax für Common Lisp, wobei sie gerade in Verbindung mit CMUCL schneller matcht als Perl selbst.

CL-WHO *with-html-output* – eine Lisp-Markupsprache, die S-Expressions in HTML übersetzt

[16] Für Informationen über Subversion siehe <URL:http://subversion.tigris.org/>
[17] Die Heap-Größe ist hart limitiert.

TBNL *To Be Named Later* – ein Toolkit, um dynamische Websites mit Common Lisp zu erstellen. Diese Bibliothek benötigt einige weitere, die auf der CD enthalten sind. Alle Bibliotheken sind wie Clark ASDF-installierbar, das heißt es reicht, einen symbolischen Link zur jeweiligen ASD-Datei in das Lisp bekannte ASDF-Verzeichnis zu setzen, und alle Bibliotheken werden automatisch bei Bedarf kompiliert und geladen.

Die letzten beiden sind nur nötig, wenn das Webinterface genutzt werden soll. Dieses wiederum benötigt den Webserver Apache in Version 1.3x mit `mod_lisp`, welches ebenfalls auf der CD zu finden ist.

CMUCL braucht nur in ein beliebiges Verzeichnis kopiert zu werden und ist einsatzbereit. Die Datei *sample-wrapper* ist ein Beispiel-Aufrufskript, in dem lediglich der Pfad zum Verzeichnis gesetzt werden muß, es kann dann unter einem beliebigen Namen – *lisp* böte sich dafür an – an einen Ort kopiert werden, der im Pfad des Benutzers liegt. Auch die ASDF-Bibliotheken können an einem beliebigen Ort liegen. Im Lisp muß ASDF geladen sein. Es liegt ein Lisp-Corefile bei, das ASDF bereits enthält. Alternativ kann man es einfach beim Start laden, dann sollte man gleich in *asdf.lisp* in die Variable `*central-registry*` den Pfad zu den ASD-Dateien eintragen.

ASDF kümmert sich selbständig um die Abhängigkeiten, das heißt, wenn Clark mit **(asdf:oos** 'asdf:load−op 'clark) geladen wird, werden automatisch die nötigen Bibliotheken dazugeladen.

Obige Anweisungen gelten dann, wenn kein Lisp auf dem System installiert ist. Unter Debian GNU/Linux reicht es, mit `apt-get install` *paketname* die Pakete automatisch zu installieren. Das gilt nicht nur fürs Basis-Lisp, sondern auch für `mod_lisp` und die meisten der erwähnten Bibliotheken. Ähnlich ist die Situation bei FreeBSD. Auch unter Mac OS X kann man den Großteil der nötigen Bestandteile für Clark mit Hilfe von Fink oder Darwinports automatisch installieren.

Wenn Clark geladen ist, ist es sinnvoll, mit **(in−package** #:clark) in seinen Namespace zu wechseln, ansonsten sind nur die wenigen exportierten Symbole direkt zugänglich.

(initialize) baut sodann die Datenstrukturen auf. Das dauert gute zehn Minuten. Solange es keine Änderungen am Programm bzw. den Daten gibt, die ein neuerliches Aufbauen nahelegen, kann man das Lisp in ein Corefile schreiben, unter CMUCL mit **(ext:save−lisp "corename")**. Wird beim Starten dieses Corefile angegeben, dauert der Start nur einige Sekunden, und man kann direkt an dem Punkt weitermachen, an dem das Corefile gespeichert wurde. Alle Funktionsdefinitionen und Daten inklusive der großen Hashtabellen sind vorhanden.

3.3.1 Arbeiten mit der IDE

Clark kann, am besten über Emacs / SLIME, direkt vom Lisp-REPL aus angesprochen werden. Das SLIME-Paket liegt der Arbeit bei. Es kann an einen beliebigen Ort kopiert werden. In der Datei *.emacs* ist dann dieser Eintrag vorzunehmen:

```
(setq load-path (append '("/pfad/zu/slime") load-path))
;;; slime
(require 'slime)
(slime-setup)
(add-hook 'lisp-mode-hook (lambda () (slime-mode t)))
(add-hook 'inferior-lisp-mode-hook (lambda () (inferior-slime-mode t)))
(setq inferior-lisp-program "lisp") ; Name des Sample-Wrappers
```

Eine Sitzung könnte beispielsweise so gestartet werden:

- Emacs starten
- Ins Clark-Codeverzeichnis wechseln: `C-x d /pfad/zu/clark/code`
- SLIME mit `M-x slime` laden
- Clark starten. Im SLIME-REPL: **(asdf:oos** 'asdf:load−op 'clark)
- In den Clark-Namespace wechseln: **(in−package** #:clark)
- Clark initialisieren: **(initialize)**

Nun kann man Clark beliebig modifizieren und alle Bestandteile testen. In *auxiliary.lisp* sind einige Komfort-Makros definiert, um bequem auf die Hashtabellen zugreifen zu können. **(ortref "81677")** ermittelt die Werte zur Postleitzahl 81677 aus der Tabelle `*orthash*`, Ähnliches gibt es für die Straßen-Tabellen.

3.3.2 Clark als Bibliothek

Clark kann nicht nur als isoliertes Programm genutzt, sondern auch als Bibliothek von anderen Lisp-Programmen angesprochen werden. Die API-Funktionen sind in *api.lisp* definiert, in *packages.lisp* steht, welche Symbole exportiert werden. Weitere API-Funktionen können in wenigen Minuten definiert werden; ihr Inhalt ist vom Einsatzzweck und Input- bzw. Outputformat abhängig.

3.3.3 Das Webinterface

Soll das Webinterface genutzt werden, muß auf dem System ein Apache in Version 1.x vorhanden sein. `mod_lisp` ist einfach zu installieren. Der Befehl `apxs -i -c mod_lisp.c` kompiliert das Modul, installiert es und macht es für Apache verfügbar.

Folgende Einträge sind in der Konfigurationsdatei von Apache nötig:

```
LoadModule lisp_module        libexec/mod_lisp.so
AddModule mod_lisp.c
# Lisp support
LispServer 127.0.0.1 3000 "lispd"
<Location /lisp>
    SetHandler lisp-handler
</Location>
```

In diesem Fall lauscht der Lisp-Server auf Port 3000, alle URLs mit `lisp/`-Präfix werden an den Lisp-Server weitergeleitet.

Das Webinterface wird mit **(start−tbnl)** gestartet, mit **(stop−tbnl)** angehalten. In der momentanen Implementierung gibt es eine paßwortgeschützte Startseite mit einem Eingabefeld für eine oder mehrere URLs, die nach Adressen abgegrast werden. Die Ergebnisseite zeigt alle gefundenen Adressen oder alternativ Links auf der Seite, die möglicherweise auf Seiten mit Adressen verweisen.

3.4 Probleme

Insbesondere die Mapping-Tabelle der Straßennamen ist sehr groß und beansprucht eine Menge Arbeitsspeicher. Nicht jedes der freien Lisps kann damit umgehen. Für CMUCL ist es nötig, die Größe des dynamisch allozierbaren Speichers zu erhöhen. Dazu ist es mit der Option `-dynamic-space-size 1664000` zu starten.

Es werden nur übliche Rechtschreibfehler erkannt. Einfache Vertipper kann das Programm nicht erkennen.

Alle Straßen und PLZ-Ort-Kombinationen, die nicht in der Datenbasis vorhanden sind, werden nicht erkannt. Es sind Funktionen implementiert, mit denen man neue Daten aufnehmen kann; diese werden sowohl in das laufende Image als auch in die Listen geschrieben: **(add−ort** liste) und **(add−street** liste) nehmen jeweils eine Liste von Strings als Argument. Bislang ist das ein manueller Prozeß, es wäre sinnvoll, die Aufnahme neuer Daten zu automatisieren.

Die Inputdaten werden zur Zeit in eine Liste von Tokens überführt. Das ist normalerweise kein Performance-Problem, könnte aber eins sein, wenn nicht linear durch die Liste gegangen wird, sondern ein Matcher wahlfreien Zugriff braucht. Es wäre daher ratsam, statt einer Liste einen Vektor zu verwenden, der auch bei wahlfreiem Zugriff eine Zeitkomplexität von $O(1)$ hat.

`Lynx` bleibt hängen, wenn der angesprochene Webserver hängt. Auch die Option `-with-timeout` verhindert das nicht in allen Fällen. Auch wenn ein Webserver kaputte Daten liefert, zum Beispiel PHP-Fehlermeldungen, reagiert `Lynx` nicht immer in wünschenswerter Weise. Clark läuft dann nicht weiter, weil er auf das Ende des Kindprozesses wartet. Dieser muß manuell beendet werden. Das Problem sollte entweder innerhalb Clarks abgefangen werden; als Workaround könnte aber auch ein Cronjob die laufenden `Lynx`-Instanzen überwachen und gegebenenfalls beenden.

4 Mögliche Erweiterungen für Clark und Ansätze, wie die Erkennungsleistung weiter verbessert werden kann

4.1 Normgerechte Formatierung und Rechtschreibkorrektur

Die gefundenen Adressen werden in dem Format ausgegeben, wie sie im Original vorlagen. Insbesondere bei Telefon- und Faxnummern gibt es zig verschiedene Möglichkeiten der Formatierung. Mit einer vorhandenen Vorwahlenliste könnten die Nummern vor der Ausgabe in ein einheitliches Format gebracht werden. Die Straßennamen-Mapping-Tabelle wird zur Zeit noch nicht vollständig genutzt. Mit ihr könnten auch Rechtschreibfehler in Straßennamen korrigiert werden.

4.2 Approximative Suche

Clark berücksichtigt zwar häufige Rechtschreibfehler in Straßennamen, scheitert aber an ungewöhnlichen Fehlern und einfachen Vertippern. Ein approximativer Suchalgorithmus wie Levenshtein[1] würde es ermöglichen, auch solche Adreßbestandteile zu erkennen. Dabei wäre zu untersuchen, ob die approximative Suche die Qualität der Suchergebnisse negativ beeinflußt.

4.3 Abgleich von Telefonnummern mit Postleitzahlen

Ein weiteres Kriterium für Zuordnung von Adreßbestandteilen und Verifikation wäre ein Abgleich von Telefonnummern und Postleitzahlen. Bei Festnetzanschlüssen kann geprüft werden, ob die Vorwahl zur angegebenen Postleitzahl paßt. Dies kann auch dazu dienen, Tippfehler in den Zahlen zu entdecken und korrigieren. Eine dafür nutzbare Liste ist bereits vorhanden, aber noch nicht ins Programm integriert.

[1] Siehe zum Beispiel <URL:http://www.merriampark.com/ld.htm>

4.4 heuristische Erkennung fehlender Adreßbestandteile

Wenn eine Adresse nur teilweise erkannt wurde, kann das auch daran liegen, daß der Datenbestand unvollständig ist. In diesem Fall könnten reguläre Ausdrücke zum Einsatz kommen, die an den Fehlstellen nachsehen, ob der entsprechende Textteil der Form nach paßt (zum Beispiel ob ein Straßengrundwort plus eventuell eine Hausnummer vorkommt; dann könnte der Teil, wenn er neben einem erkannten Ortsnamen steht, als Straßenkandidat angesehen werden). Diese Kandidaten könnten, entsprechend gekennzeichnet, der Adresse zugewiesen werden. Darüberhinaus könnte man sie in einen „Quarantäne-Datenbestand“ aufnehmen, der nach Prüfung in die tatsächlichen Programmdaten übergehen kann.

Neben regulären Ausdrücken könnten heuristische Erkenner insbesondere bei Adreßlisten auch HTML-Skelette oder ähnliche Umfeldinformationen nutzen (siehe Abschnitt 1.6.2).

4.5 Verbesserung der Namenserkennung

Die Namenserkennung funktioniert in der derzeitigen Implementierung nur schlecht; sie war auch kein hauptsächliches Entwicklungsziel. Für künftige Erweiterungen des Programms wäre es wünschenswert, diesen Teil des Programms stark zu erweitern. Dazu sind bessere (sprich: deutlich kompliziertere) Algorithmen ebenso notwendig wie entsprechende Wortlisten.

4.6 Automatische Erweiterung des Datenbestandes

Wie oben bereits angemerkt, können Heuristiken, die fehlende Adreßbestandteile ergänzen, dazu dienlich sein, den Datenbestand automatisch zu erweitern. Trotzdem wird es periodisch notwendig sein, die neu gewonnenen Daten manuell zu verifizieren.

4.7 Kongruenz URL/E-Mail, statistische Methoden zum Finden der richtigen Adresse

Clark gibt momentan alle Adressen aus, die er auf einer Seite findet. Gerade beim Extrahieren von Adressen aus Webseiten ergibt sich dabei das Problem, daß auch unerwünschte Adressen ausgegeben werden, zum Beispiel die des Webdesigners. Um die gewünschte Adresse als einzige oder erste auszugeben, sind verschiedene Mechanismen denkbar. So könnten E-Mail-Adressen mit dem Namen der Domain abgeglichen werden und die dazugehörige Anschrift eine höhere Relevanz bekommen, wenn Domain und Domainteil der E-Mail-Adresse übereinstimmen.

Ebenso könnte man statistische Methoden einsetzen. Dann wäre es allerdings auch notwendig, nicht nur die Seite mit der Adresse zu betrachten, sondern der ganze Site müßte ausgewertet werden; Domainname und textueller Inhalt könnten auf eine Häufung bestimmmter Namen untersucht werden, welche man dann mit den gefundenen Adressen abgleichen könnte – oder sie sogar dazu heranziehen, die Namenserkennung zu verbessern, indem man diese Häufungen vor der Namenserkennung untersucht und sie dabei berücksichtigt.

5 Partielle Auswertung und Fazit

5.1 Auswertung eines Testlaufs

Die folgenden Daten beziehen sich auf einen Programmlauf, dessen kompletter Output der Arbeit beiliegt. Clark bekam eine URL-Liste mit 43.711 Einträgen als Input, wobei zum Zeitpunkt der Erstellung dieser Liste hinter jedem URL eine Webseite mit einer Adresse zu finden war.

Die Berechnung von Präzision und Recall erfolgte auf Basis von 100 zufällig aus dem Output herausgegriffenen Suchergebnissen. Es wurde dabei händisch überprüft, ob die Originalangaben noch stimmen, oder ob sich die Seite seit der Erstellung der URL-Liste verändert hat. Außerdem wurde überprüft, ob Clark alle Adressen erkannt hat. Die Namenserkennung wurde nicht berücksichtigt, sie ist getrennt von den anderen Ergebnissen angegeben. Als erkannt gilt eine Adresse, wenn der Anschriftbestandteil und, falls vorhanden, Festnetztelefon- und Faxnummern erkannt wurden. Nicht-Erkennung wegen Vertippern in Adressen wurde nicht gewertet, da das Programm in der derzeitigen Version diese prinzipbedingt nicht erkennen kann.

In den 100 Webseiten fanden sich 147 Adressen. Davon wurden 139 gefunden. Zu 78 dieser 139 gefundenen Adressen wurden die Namen erkannt. Zwei Adreßbestandteile wurden falsch erkannt: eine Telefonnummer, die eine Telefongebührenangabe war, und eine Straße „Werbeplatz 62“, die am Fundort keine war. Daraus ergibt sich eine Präzision von 98,6 % und ein Recall von 94,6 %. Der Recall für Namen liegt bei 53,1%. Nur die falsche Telefonnummer wurde in eine Anschrift integriert, der Werbeplatz als überschüssig ausgegeben. Wenn man dies noch berücksichtigte, ergäbe sich sogar eine noch höhere Präzision.

Zu den nicht gefundenen Adressen ist zu sagen, daß eine Adresse auch dann als nicht gefunden galt, wenn nur die vorhandene Telefonnummer nicht erkannt wurde. Ansonsten wäre der Wert höher gewesen.

5.2 Fazit

Das Programm funktioniert zum gegenwärtigen Zeitpunkt schon recht gut, wenn auch der eine oder andere Sonderfall noch in die Matching-Algorithmen integriert werden muß. Ein Manko ist die Namenserkennung, jedoch bin ich zuversichtlich, daß man auch dieses

Problem durch eine Kombination aus Wortlisten, Statistik und schlauen Heuristiken in den Griff bekommen kann. Die Codebasis von Clark bietet dafür gute Voraussetzungen.

Die Arbeit hat gezeigt, wie viel Aufwand es schon bedeutet, nur einen so kleinen Teil der Sprache wie Adreßinformationen automatisiert erkennen und verarbeiten zu können. Sie hat aber auch gezeigt, daß es durchaus machbar ist, wenn man den Aufwand nicht scheut.

Literaturverzeichnis

[ADO1974] ADO – Allgemeine Dienstordung mit DIN-Norm 5008, 13. Auflage 1974, Verlag für Verwaltungspraxis Franz Rehm KG, München

[Hovermann] Eike Hovermann, Bürokommunikation – Schreiben und Gestalten nach DIN 5008, Reihe DataBook, CD-ROM, 1. Auflage 1996, Rossipaul Medien GmbH, München

[Duden] DUDEN-Redaktion (Hrsg.), DUDEN „Rechtschreibung der deutschen Sprache und der Fremdwörter“, 19., neu bearbeitete und erweiterte Auflage 1986, Bibliographisches Institut Mannheim

[BVA2004] Bundesverwaltungsamt, DIN 5008: Neuregelung des Anschriftenfeldes (INFO 1815, Juli 2004), im WWW am 12.09.2004 unter <URL:http://www.bva.bund.de/aufgaben/win/beitraege/00302/>, PDF-Dokument liegt der Arbeit bei.

[Parnas] Dagfinn Parnas, How to cope with incorrect HTML, wissenschaftliche Abschlußarbeit an der Universität Bergen im Jahr 2001, zum Download unter <URL:http://www.ub.uib.no/elpub/2001/h/413001/Hovedoppgave.pdf>, liegt der Arbeit bei

[Ziegler] Cai Ziegler, Gekelterte Semantik – Datenextraktion im Web, in iX – Magazin für professionelle Informationstechnik, Ausgabe 07/2004, Heise Zeitschriften Verlag GmbH & Co. KG, Hannover

[Altfeld] Jürgen Altfeld, Maschineller Adressen-Abgleich – Extraktion und Struktur-Analyse von Adressen aus Web-Seiten, Magisterarbeit an der Ludwig-Maximilians-Universität München, Centrum für Informations- und Sprachverarbeitung, im Jahr 2000, Bibliothek der Institute am Englischen Garten

[Lubenow] Joe Lubenow, Universal Postal Union Publishes UPU S42 International Address Standard, im WWW am 21.09.2004 unter <URL:http://xml.coverpages.org/ni2003-06-17-a.html>, PDF-Version liegt der Arbeit bei

[PATDL] Joe Lubenow, Postal Address Template Description Language (PATDL): A Contribution To An International Postal Addressing Standard, im WWW am

21.09.2004 unter <URL:http://xml.coverpages.org/Lubenow-PATDL200204.pdf>, liegt der Arbeit bei

[antiword] Homepage (23.09.2004): <URL:http://www.winfield.demon.nl/>

[pdftotext] Homepage (23.09.2004): <URL:http://www.foolabs.com/xpdf/>

[Lynx] Homepage (23.09.2004): <URL:http://lynx.browser.org/>

[w3m] Homepage (24.09.2004): <URL:http://w3m.sourceforge.net/>

[Friedl1] regulärer Ausdruck für syntaktisch korrekte E-Mail-Adressen nach RFC822, aus: Jeffrey Friedl, Mastering Regular Expressions, 1997, O'Reilly & Associates, Inc., Sebastopol, CA, USA, im WWW am 24.09.2004 unter <URL:http://examples.oreilly.com/regex/email-opt.pl>

[Friedl2] Jeffrey Friedl, Mastering Regular Expressions, Second Edition, 2002, O'Reilly & Associates, Inc., Sebastopol, CA, USA

[RFC822] Standard for the Format of ARPA Internet Text Messages, zum Download unter <URL:ftp://ftp.rfc-editor.org/in-notes/rfc822.txt>

[RFC2822] Standard for the Format of ARPA Internet Text Messages, zum Download unter <URL:ftp://ftp.rfc-editor.org/in-notes/rfc2822.txt>

[screen] Homepage (24.09.2004): <URL:http://www.gnu.org/software/screen/>

[SLIME] Superior Lisp Interaction Mode for Emacs, Homepage (24.09.2004): <URL:http://common-lisp.net/project/slime/>

[CMUCL] Carnegie-Mellon University Common Lisp, Homepage (24.09.2004): <URL:http://www.cons.org/cmucl/>

[ASDF] Another System Definition Facility, Hinweise dazu im Common Lisp Wiki (24.09.2004): <URL:http://www.cliki.net/asdf>

[Weitz] Dr. Edmund Weitz, verschiedene Programmbibliotheken für Common Lisp, Homepage (24.09.2004): <URL:http://www.weitz.de/>

A Lebenslauf

Persönliche Daten

Wolfgang Alexander Mederle
Prinzregentenstr. 124
81677 München

Tel.: (089) 41 90 16 80
E-Mail: wolfgang ät mederle.de

Geb. am 23. 04. 1972 in München
Verheiratet seit 10.2000 mit So-Young Kim
deutsche Staatsangehörigkeit

Schulbildung

1978-1982	Grundschule an der Senftenauerstraße, München
1982-1989	Donau-Gymnasium Kelheim
1989-1991	Michaeli-Gymnasium München

Zivildienst

09/1991–02/1993	Erste-Hilfe-Ausbilder und Rettungsdiensthelfer beim Malteser-Hilfsdienst München

Berufsausbildung

09/1993–06/1996	Automobilmechaniker (IHK) bei der Mercedes-Benz AG, Niederlassung München

Studium

10/1996–09/1999	Diplomstudiengang Betriebswirtschaftslehre an der LMU München
10/1999–dato	Magisterstudiengang Computerlinguistik mit Informatik und Betriebswirtschaftslehre an der LMU

Berufserfahrung und sonstige Tätigkeiten

03/1993–09/1993	ehren- und nebenamtliche Tätigkeit beim Malteser-Hilfsdienst als Ausbilder für Erst- und Sanitätshelfer, Spezialgebiet Erste Hilfe bei

	Notfällen mit Säuglingen und Kleinkindern; Kursorganisation und Werbung, daneben Weiterbildung zum Rettungssanitäter
06/1996–09/1996	befristete Anstellung bei Mercedes-Benz in München, Aufgabenstellung: Inventur über die gesamte Schließanlage aller Niederlassungsfilialen, Anpassen der vorhandenen Grundrißpläne, Installation, Konfiguration und Dokumentation eines Datenverwaltungsprogramms dafür
04/1997–06/2000	Call-Center-Agent bei Mercur Assistance Deutschland GmbH, Bereich Unfallschadenerstaufnahme, Schadensteuerung, Schutzbriefleistungsabwicklung
07/2000–07/2001	Secunet Security Networks GmbH, Schulungszentrum; Netzwerkadministration am Schulungszentrum, Kursvorbereitung
07/2001–dato	studentische Hilfskraft am Centrum für Informations- und Sprachverarbeitung an der LMU; freiberufliche Tätigkeit als Netzwerkbetreuer und Ausbilder im Bereich Betriebssysteme und Netzwerksicherheit/Internet

Fremdsprachen

Englisch, Spanisch

EDV

Betriebssysteme	UNIX: Linux (diverse Distributionen), Mac OS X, NeXTSTEP, OpenSTEP, FreeBSD, Tru64; andere: Windows, Novell Netware, DOS
Sprachen	Common Lisp, Emacs Lisp, Bash, AppleScript, Java, Perl, SQL, m4, C, C++, Objective-C, Prolog, CHR, HTML, XML, JavaScript, BASIC
Anwendungen	Emacs, LaTeX, StarOffice/OpenOffice, Microsoft Office incl. Access, Dreamweaver, Paintshop Pro, GIMP, Xcode, Keynote, Unitex, RagTime, Apache, PostgreSQL, MySQL, postfix etc.

Interessen Computer und Internet, Oldtimer, Geschichte des Automobilbaus, Sprache und Kommunikation, Literatur, Design, Musik, Spanien und Korea

München, 27. September 2004

B Erklärung zur Magisterarbeit

LUDWIG MAXIMILIANS UNIVERSITÄT
CENTRUM FÜR INFORMATIONS- UND SPRACHVERARBEITUNG

Hiermit versichere ich, dass die vorgelegte Magisterarbeit selbständig verfaßt und noch nicht anderweitig zu Prüfungszwecken vorgelegt wurde. Alle benutzen Quellen und Hilfsmittel sind angegebene, wörtliche und sinngemäße Zitate und wurden als solche gekennzeichnet.

München, den ....