CLab

Web-basiertes
virtuelles Laboratorium zur Computerlinguistik

LERNEINHEIT

Verwendung regulärer Ausdrücke

Universität Zürich
Institut für Computerlinguistik

Prof. Dr. Michael Hess

---

1. Voraussetzungen

Bevor Sie diese Lerneinheit bearbeiten, sollten Sie wissen, was Trunkierung und ‘‘Stemming’’ ist. Dazu siehe ⇒ Skript ECL 1.

2. Lernziel

Sie sollten umfassend Bescheid wissen, was reguläre Ausdrücke sind und wozu man sie einsetzen kann. Sie sollten auch im Stande sein, selbst reguläre Ausdrücke nach vorgegebenen Kriterien zu bilden.

Wozu braucht man RA?

Die Verwendung regulärer Ausdrücke (RA) ist heute weit verbreitet im Bereich grundlegender, einfacherer Verfahren der Computerlinguistik (z.B. Tokenisierung und Tagging), der sogenannten ‘‘flachen Textverarbeitung’’.

Mit Hilfe von regulären Ausdrücken lassen sich *Muster* für einfache sprachliche Ausdrücke spezifizieren, nach denen automatisch in Texten gesucht werden kann. Ein Beispiel hierfür, umgangssprachlich ausgedrückt, wäre ‘‘Wort, das mit einem Grossbuchstaben anfängt und mit einem ’k’ aufhört’’. Je nach Textkorpus, in dem man sucht, lässt sich damit das Wort ‘‘Computerlinguistik’’ finden. Die gefundenen Instanzen (‘‘Matches’’) eines solchen Mustervergleichs (‘‘Pattern matching’’) stehen dann zum Teil direkt der weiteren Verarbeitung zur Verfügung.

Woher kommen RA?

Die Verwendung regulärer Ausdrücke ist vor allem durch Suchfunktionen (*egrep*) und Skriptsprachen (*lex* und *flex*) auf Unix-Ebene sowie durch plattformübergreifende Skriptsprachen (vor allem *Perl*) bekannt geworden[1] Mittlerweile können RA in vielen gängigen Programmiersprachen verwendet werden.

Was sind RA?

Reguläre Ausdrücke sind zunächst einmal einfach zu spezifizieren. ‘‘Einfach’’ heisst dabei nicht unbedingt ‘‘leicht verständlich’’, sondern ‘‘gemäss formaler Kriterien wie Repräsentations- und Verarbeitungskomplexität einfach’’ (das macht sie

1. Man glaube aber ja nicht, dass es keine Verwirrung stiftenden Unterschiede in diesem Bereich gibt. Nachzulesen in einem informativen Artikel zu ⇒Mustersuche und Reguläre Ausdrücke.

so attraktiv). Entsprechend lassen sich Sprachen, die RA beschreiben (sogenannte reguläre Sprachen), mithilfe des ‘‘einfachsten’’ Grammatiktyps (sogenannten Typ-3-Grammatiken) beschreiben. Solche Sprachen können nachweislich in effizient zu verarbeitende Strukturen (sogenannte Endliche Automaten) übersetzt werden, die durch RA spezifizierte Ausdrücke schnell aufspüren. [2]

Was kann man mit RA machen?

Programmiersprachen, die die Verwendung von RA implementieren, bieten in der Regel mindestens die folgenden Möglichkeiten:

- checken, *ob* ein RA in einem Text vorkommt
- angeben, *wo* ein RA in einem Text vorkommt
- den *Zugriff auf Teile eines Matches* im Text
- den *Zugriff auf alle zu findenden Matches* im Text
- die Möglichkeit zur *Ersetzung* eines (oder aller) Vorkommen in einem Text durch etwas anderes
- die Möglichkeit zur *Zerlegung* eines Textes an den Stellen, an denen ein gegebener RA vorkommt.

⇒**QUIZ:** Reguläre Ausdrücke: Eigenschaften

2. Diese Termini werden an ⇒anderer Stelle sehr viel ausführlicher erklärt.

3. Konstruktion regulärer Ausdrücke

Wie ‘‘baut’’ man RA?

Im folgenden sollen die wesentlichen Möglichkeiten zur Konstruktion von RA vorgestellt werden. Hierzu werden RA in den Beispielen stets durch Schrägstriche ("/") (‘‘slashes’’) eingerahmt (wie es z.B. in Perl üblich ist). Anhand eines solchermassen angegebenen Suchmusters wird ein Text (hier: ‘‘Jeder Mensch ist einzigartig und besitzt eine eigene spezifische Persönlichkeit’’) von Anfang bis Ende durchsucht und es wird der zuerst gefundene Match als Ergebnis geliefert. Im Folgenden gehen wir davon aus, dass jeweils *alle* Ergebnisse aufgesammelt werden.

3.1 Direkte Angabe von Zeichenketten

Direkte Angabe von **Zeichenketten**, also Abfolgen von Zeichen, sind der einfachste Fall:

```
/einzig/
```

Suche nach dem Vorkommen von "einzig" im Text

"Jeder Mensch ist **einzig**artig und besitzt eine eigene spezifische Persönlichkeit"

Matches: ["einzig"]

Hierzu finden Sie hier eine
⇒**ILAP** *(Übung):* Pattern Matching (Zeichenketten)

3.2 Zeichenklassen

Zeichenklassen ("character classes") sind eine wichtige Möglichkeit, reguläre Ausdrücke kürzer zu formulieren: In "[]" eingeschlossen lassen sich Zeichen angeben, die an einer bestimmten Position auftreten sollen/dürfen. Sie dienen im wesentlichen der Vereinfachung von RA (um nicht jeweils die Zeichenalternativen an einer Position explizit angeben zu müssen).

```
/[MP]../
```

Suche nach dem Vorkommen von "M" oder "P" gefolgt von zwei beliebigen Zeichen

"Jeder **Men**sch ist einzigartig und besitzt eine eigene spezifische **Per**sönlichkeit"

Matches: ["Men", "Per"]

weitere Möglichkeiten, Zeichenklassen zu spezifizieren:

- `/[A-Z]/` ein beliebiger Grossbuchstabe (aber ohne Umlaute, ß und Buchstaben mit Akzenten)
- `/[A-Za-z]/` ein beliebiger Gross- oder Kleinbuchstabe (ditto)
- `/[0-9]/` eine beliebige Ziffer
- Mischungen und Teilbereichsangaben sind möglich, z.B. `/[ab17G-K]/`
-

3.3 Vorgefertigte Zeichenklassen und Metazeichen

Vorgefertigte Zeichenklassen sind **besondere und Sonder-Zeichen:** Es gibt eine Reihe von Zeichen, die angelehnt an die Textverarbeitung in Programmiersprachen wie folgt gematcht werden können:

- `/\t/` Tabulator
- `/\n/` newline
- `/\r/` return

- `/\f/` form feed
- `/\w/` alphanumerisches Zeichen (‘‘word character’’), entspricht dem längeren RA: `/[A-Za-z0-9_]/` und auch dem RA `/[:alpha:]/`
- `/\W/` nicht-alphanumerisches Zeichen, entspricht dem längeren RA: `/[^A-Za-z0-9_]/`
- `/\s/` whitespace character, entspricht dem längeren RA: `/[\f\n\r\t\v]/`
- `/\S/` non-whitespace character, entspricht dem längeren RA: `/[^\f\n\r\t\v]/`
- `/[:blank:]/` space oder Tabulator
- `/\d/` Ziffer, entspricht dem längeren RA: `/[0-9]/` und auch dem RA `/[:digit:]/`
- `/\D/` Nicht-Ziffer, entspricht dem längeren RA: `/[^0-9]/`
- `/\b/` Wortgrenze (d.h., Grenze zwischen `\w` und `\W` bzw. `\W` und `\w`)
- `/\B/` Nicht-Wortgrenze
- `/\x00/` Hexadezimal
- `/\</` Beginn eines Wortes
- `/\>/` Ende eines Wortes

Für einige Zeichenklassen gibt es mehrere Möglichkeiten, einen entsprechenden RA zu formulieren.

Metazeichen (‘‘metacharacters’’) sind Zeichen, die eine besondere Funktion in einem RA haben. Der wichtigste ist der Punkt ("."), der auf alle Zeichen (ausser dem carriage return) matcht.

```
/.ei./
```

Suche nach dem Vorkommen von "ei" nach und vor einem beliebigen Zeichen

"Jeder Mensch ist **ein**zigartig und besitzt **ein**e **eig**ene spezifische Persönlich**keit**"

Matches: [" ein", " ein", " eig", "keit"]

Hierzu finden Sie hier eine
⇒**ILAP** *(Übung):* Pattern Matching

3.4 Verankerungen

Verankerungen (*anchors*) definieren einen festen Punkt innerhalb der Zeile, an dem das Suchmuster festgemacht wird:

- `/^/` der Anfang einer Zeile (ausser innerhalb einer Zeichenklasse, dort in der Bedeutung ‘bilde das Komplement zu’)
- `/$/` das Ende einer Zeile
- `\` ("Backslash"): Einem Metazeichen vorangestellt, wird dieses selbst gematcht (d.h., mit `/\./` findet man Punkte im Text)
- `|`, `(`, `)`, `[`, `]`: sind ebenfalls Metazeichen. `|` entspricht einem ‘‘oder’’ im RA (Alternative), `()` gruppieren und `[]` geben eine Zeichenklasse an.

Wichtig zu wissen

Hierzu finden Sie hier eine
⇒**ILAP** *(Übung):* Pattern Matching (Verankerung)

3.5 Optionalität, Gruppierung, Disjunktion

Alternativen werden durch `|` getrennt und am besten von runden Klammern `()` eingeschlossen. Auf diese Weise kann eine Disjunktion in einen grösseren Ausdruck eingebettet werden.[3] Nur bei einfachen Mustern reicht die Angabe der Disjunktion selbst.

```
/(einzig|spezifisch)/
```

oder

```
/einzig|spezifisch/
```

3. Insbesondere hat der Disjunktionsoperator eine geringe Präzedenz, so dass die Disjunktion entsprechend abgeschlossen werden muss.

Suche nach dem Vorkommen von "einzig" oder "spezifisch" im Text

"Jeder Mensch ist **einzig**artig und besitzt eine eigene **spezifisch**e Persönlichkeit"

Matches: ["einzig", "spezifisch"]

Hierzu finden Sie hier eine
⇒**ILAP** *(Übung):* Pattern Matching (Optionalität, Gruppierung, Disjunktion)

3.6 Allgemeine Wiederholungen

Quantifikatoren geben an, wie häufig das direkt vorangehende Zeichen bzw. die direkt vorangehende Gruppe oder Zeichenklasse vorkommen können soll. Es gibt einige allgemeine und eine Reihe von numerischen Wiederholungen.

Allgemeine Wiederholungen sind:

- `*` (‘‘Kleene-Stern’’) Keinmal oder beliebig oft
- `+` (‘‘Kleene-Plus’’) Mindestens einmal
- `?` Einmal oder keinmal

Hierzu finden Sie hier eine
⇒**ILAP** *(Übung):* Pattern Matching (Wiederholungen)

3.7 Numerische Wiederholungen

Numerische Wiederholungen sind:

- `{n}` n-mal
- `{n,}` mindestens n-mal
- `{n,m}` mindestens n-mal, aber höchstens m-mal
- das heisst: `*` entspricht `{0,}`, `+` entspricht `{1,}` und `?` entspricht `{0,1}`.

```
/ei.{0,2}ne/
```

Suche nach dem Vorkommen von "ei" am Anfang und "ne" am Ende einer Zeichenkette, die insgesamt 4 bis 6 Zeichen lang ist

"Jeder Mensch ist einzigartig und besitzt **eine eigene** spezifische Persönlichkeit"

Matches: ["eine", "eigene"]

wichtig

Die Quantifikatoren matchen per default immer so oft wie möglich, d.h. das Teilmuster findet den längstmöglichen Match (man spricht hier auch von ‘‘gefrässigen’’[4] Mustern). Deshalb bedeuten

```
/ei.{0,2}ne/
```

von oben und

```
/ei(..)?ne/
```

nicht das gleiche: Beim zweiten Ausdruck sind nur "eine" (also 4 Zeichen) oder "eiXXne" (also 6 Zeichen, mit X ein beliebiges Zeichen) möglich.

Wünscht man hingegen den jeweils kürzesten Match, so muss den Quantifikatoren ein "?" angehängt werden.

```
/\b[egin]{3,3}.*\b/
```

Suche nach dem Vorkommen einer Zeichenkette, die mit einer Wortgrenze beginnt, gefolgt von genau 3 Zeichen, die jeweils entweder "e", "g", "i" oder "n" sind, gefolgt von beliebigen Zeichen bis zu einer Wortgrenze.

"Jeder Mensch ist **einzigartig und besitzt eine eigene spezifische Persönlichkeit**"

Matches: ["einzigartig und besitzt eine eigene spezifische Persönlichkeit"]

```
/\b[egin]{3,3}.*?\b/
```

4. Englisch "greedy".

Suche nach dem Vorkommen einer Zeichenkette, die mit einer Wortgrenze beginnt, gefolgt von genau 3 Zeichen, die jeweils entweder "e", "g", "i" oder "n" sind, gefolgt von beliebigen Zeichen bis zur ersten Wortgrenze.

"Jeder Mensch ist **einzigartig** und besitzt **eine eigene** spezifische Persönlichkeit"

Matches: ["einzigartig", "eine", "eigene"]

Hierzu finden Sie hier eine
⇒**ILAP** *(Übung):* Pattern Matching (Numerische Wiederholungen)

3.8 Ersetzen und Rückverweise

Rückverweise sind ein sehr mächtiges Instrument beim Schreiben von regulären Ausdrücken. Für gruppierte (d.h. in runde Klammern eingeschlossene) Teilmuster werden systematisch Register angelegt, so dass mit Hilfe von Variablen der Form "$i" (mit i aus [1-9]) ein Rückbezug auf Matches dieser Teilstrukturen möglich ist. Dies ist insbesondere bei Ersetzungen sehr hilfreich.

Innerhalb des RA kann mit "\i" auf Teilmuster(matches) zurückverwiesen werden:

```
/\b([egin]{3,3}).*?\b\s\1/
```

Suche nach dem Vorkommen einer Zeichenkette, die mit einer Wortgrenze beginnt, gefolgt von genau 3 Zeichen, die jeweils entweder "e", "g", "i" oder "n" sind (dieses Muster wird als erstes Muster gespeichert), gefolgt von beliebigen Zeichen bis zur ersten Wortgrenze, gefolgt von einem white space, gefolgt vom ersten Muster.

"Jeder Mensch ist **einzigartig und besitzt ein**e eigene spezifische Persönlichkeit"

Matches: ["einzigartig und besitzt ein"]

3.9 Textsuche und Ersetzen

Optionen bei einem Mustervergleich

In Perl ist die Begrenzung des RA ("//") tatsächlich ein Operator, der einen RA auf einen Text anwendet. Gleichwertig wird der Operator `m//` verwendet. Mit beiden kann nach dem Vorkommen einer Zeichenkette gesucht werden, die durch den RA spezifiziert wird.

In Perl existiert eine einfach zu handhabende Verwendung der RA für **Ersetzungen** innerhalb eines Textes (andere Sprachen bieten ähnliche Möglichkeiten). Hier wird der Operator `s///` verwendet:

`s/`*RA*`/`*Ersetzungsausdruck*`/`

```
s/spezifisch(.*?)\b/ganz besonder$1/
```

Suche nach dem Vorkommen einer Zeichenkette, die mit ‘‘spezifisch’’ beginnt und mit beliebigen Zeichen bis zur nächsten Wortgrenze fortgesetzt wird (diese Zeichen werden in der Variable $1 gespeichert). Ersetze dies durch ‘‘ganz besonder’’ gefolgt von den gespeicherten Zeichen aus $1.

"Jeder Mensch ist einzigartig und besitzt eine eigene **ganz besondere** Persönlichkeit"

Sowohl für die Suche nach einem Muster als auch für das Ersetzen einer Zeichenkette können verschiedene Optionen übergeben werden, die Folgendes spezifizieren:

- `g` Suche nach allen Vorkommen des RA nicht nur nach dem ersten.
- `i` Suche ohne Unterscheidung von Gross-/Kleinschreibung.
- `m` Die Verankerungen ^ und $ matchen nicht bloss am Beginn und Ende der ganzen Zeichenkette, sondern auch an allen Zeilenenden in der Zeichenkette drin.
- `o` Das angegebene Muster wird nur einmal eingelesen - damit können keine Variablen im Muster verwendet werden.
- `s` Die Zeichenklasse . matcht auch Zeilenenden - damit können Muster einfacher notiert werden, welche über Zeilenenden hinweg matchen sollen.

Die Optionen werden jeweils an einen Operator ohne Leerraum angefügt. Es können mehrere Optionen angegeben werden.

```
s/spezifisch(.*?)\b/ganz besonder$1/gi
```

Suche nach dem Vorkommen einer Zeichenkette, die mit ‘‘spezifisch’’ beginnt und mit beliebigen Zeichen bis zur nächsten Wortgrenze fortgesetzt wird (diese Zeichen werden in der Variable $1 gespeichert). Ersetze dies durch ‘‘ganz besonder’’ gefolgt von den gespeicherten Zeichen aus $1. Führe die Suche bis zum Ende des abzusuchenden Textes durch. Unterscheide nicht, ob die Matches gross oder klein geschrieben sind.

Hierzu finden Sie hier eine
⇒**ILAP** *(Übung):* Pattern Matching (Ersetzen mit regulären Ausdrücken)

und zwei Selbsttests:

⇒**QUIZ:** Reguläre Ausdrücke: Basics

⇒**QUIZ:** Reguläre Ausdrücke: Match

Hier findet sich nochmals die Übersicht aller vorstehend verwendeten Übungen zu den einzelnen Typen Regulärer Ausdrücke:
⇒**ILAP** *(Übung):* Praktische Einführung "Pattern Matching"

Hier findet sich ein Experiment, in dem man die verschiedenen Typen Regulärer Ausdrücke *kombiniert* auf einen ganzen *Text* ansetzen kann:

⇒**ILAP** (*Experiment):* Pattern Matching (Suchmustern allgemein)

4. Verwendung von regulären Ausdrücken in sprachtechnologischen Systemen

Reguläre Ausdrücke finden seit einiger Zeit verstärkt Verwendung bei der ‘‘flachen’’ Textverarbeitung. Hierbei kann es sich um eigenständige Anwendungen handeln (z.B. Filterung von Meldungen) oder um die Vorverarbeitung von Texten für komplexere Aufgaben. Zum zweiten gehören insbesondere Tokenisierung (aus welchen Basiseinheiten besteht der Text?), Tagging (welche Wortart kann jeder einzelnen Basiseinheit zugewiesen werden?) und Chunk Parsing (welche Satzteile können auf einfache Weise im Text identifiziert werden?).

Wofür werden RA in der CL verwendet?

In diesen Bereichen geht es im Wesentlichen darum, einfache Operationen auf Texten auszuführen (Erkennen und Ersetzen anhand einfacher Muster), und zwar mit möglichst einfachen, allgemeinen und wiederverwendbaren Verfahren. RA bieten sich hierfür an.

Eine wesentliche Aufgabe besteht darin, Textbestandteile zu identifizieren und zu isolieren, die jeweils eine bestimmte Funktion erfüllen, z.B. die Angabe von Name, Zeit, Datum. Jede dieser einzelnen Angaben weist bestimmte strukturelle Merkmale auf, die für ihre Erkennung verwendet werden können, die man aber ungern einem allgemeinen Verfahren (z.B. dem Parser) überlassen möchte.

4.1 ‘‘Named Entity Recognition’’

Eine typische Aufgabe der Computerlinguistik, bei der man reguläre Ausdrücke verwenden kann, ist das Erkennen von sog. ‘‘benannten Einheiten’’.

Das Problem:

1. Typischerweise kann man auf das gleiche Objekt in der Welt auf verschiedenste Arten Bezug nehmen. Besonders gross ist die Variabilität offenbar bei *Bezugnahmen auf einmalige Objekte*.

   Natürlich sollte ein sprachverarbeitendes System diese Bezugnahmen als synonym erkennen und sie auf eine kanonische Form abbilden.

2. Sehr oft bestehen diese Bezugnahme auch aus mehreren Tokens. Wenn man diese Tokens nicht als zusammengehörig erkennt, wird der Parser versuchen, sie syntaktisch als Konstituente des Satzes zu erkennen.

Das ist aber sinnlos, da sie syntaktisch in aller Regel die Rolle eines Eigennamens (und damit einer NP) spielen, unabhängig von ihrer Strukturierung, und da sie immer die gleiche Bedeutung haben.

Eine Erkennung und Isolierung solcher Ausdrücke in einer frühen Phase der Analyse bedeutet also ein Problem weniger in der weiteren Verarbeitung.

Dieser Prozess wird in der Literatur irreführenderweise meist als ‘‘named entity recognition’’ bezeichnet. Aber natürlich erkennt man nicht die benannten Einheiten selbst (die Objekte in der Welt), sondern ihre Benennungen in der Sprache. Wenn schon müsste man von ‘‘named entity name recognition’’ sprechen. Besser wäre wohl eine Bezeichnung wie ‘‘Erkennung eigennamenähnlicher Ausdrücke’’ od. ä.

Zu den diesen Ausdrücken zählt man üblicherweise:

1. eigentliche Eigennamen (für Personen, Organisationen, geographische Objekte)

   Beispiele:

   - {"Harry Schearer", "H. Schearer", "Mr. Harry Schearer"}, "Secretary Robert Mosbacher", "John Doe, Jr."
   - {"Hyundai of Korea, Inc.", "Hyundai, Inc. of Korea"}, "Bridgestone Sports Co.", "University of California in Los Angeles"
   - {"European Community", "EU", "The Union"}, "NASDAQ", "Labor Party", "Trinity Lutheran Church"

2. Zeitbezeichnungen

   Beispiele: {"twelve o’clock noon", "12 am", "12:00", "12.00", "12 o’clock"}, "5 p.m. EST", "fiscal 1989", "third quarter of 1991" , "the first half of fiscal 1990"

   Ein besonders häufiges Beispiel sind Datumsangaben wie z.B. die folgenden:

   - "8. August 2001"
   - "08.08.01"
   - "8. Aug. 01"

3. numerische Massangaben (Prozent-, Temperatur-, Geldbetragsangaben)

   Beispiele: "20 million New Pesos", "$42.1 million", "15 pct", "£26 million ($43.6 million)"

Wie die Beispiele zeigen, liegt ein besonderes Problem beim Erkennen dieser Ausdrücke darin, dass diese Ausdrücke besonders oft ‘‘hässliche’’ Sonderzeichen enthalten, der bei der Tokenisierung besondere Schwierigkeiten bereitet (siehe die Lerneinheit zur ⇒Tokenisierung .

Hierzu ein
⇒**ILAP** (*Experiment*) Datumserkennung und ein

⇒**QUIZ:** Datumserkennung mit Regulären Ausdrücken

Zur spezifischen Problematik der Erkennung von Eigennamen finden Sie hier eine ⇒Lizentiatsarbeit: Jeannette Roth: Der Stand der Kunst in der Eigennamen-Erkennung. Mit einem Fokus auf Produktnamen-Erkennung; Lizentiatsarbeit; Dezember 2002.

Neben der ‘‘Named Entity Recognition’’, welche primär als eine Komponente umfassenderer Anwendungen nützlich ist, gibt es eine Reihe von Fällen, wo reguläre Ausdrücke in eigenständigen Anwendungen verwendet werden. In allen diesen Anwendungen kann man durch den sorgfältigen (und mengenmässig massiven!) Einsatz regulärer Ausdrücke recht schnell einen ersten (trügerischen) Eindruck von Sprachverstehen erzielen. Drei derartige Anwendungen von regulären Ausdrücken sollen betrachtet werden:

4.2 Filtern und Sortieren von Meldungen

Eine bekannte derartige Anwendung ist das Filtern und Sortieren von verschiedenen Arten von Texten:

- Ein erstes Beispiel dafür sind Mail-Filter-Systeme. Sie sollen bestimmte Mail-Meldungen ganz unterdrücken (z.B. Spam) und/oder nach bestimmten Kriterien in verschiedene ‘‘In-Boxes’’ des Mail-Readers leiten (sie also sortieren, oder ‘‘routen’’).
- Eine anderes Beispiel sind Systeme, welche Sammlungen von Meldungen (z.B. Newslists, Meldungsströme von Nachrichtenagenturen) nach bestimmtem Meldungen (z.B. Stellenanzeigen, Konferenzankündigungen) durchsuchen und diese extrahieren.

Die Aufgabe ist in beiden Fällen die gleiche: Auf Grund von einfachen Mustern schnell eine grosse Anzahl von Texten bearbeiten. Sie finden im folgenden eine ILAP, in der Meldungen in Newslists durch den Einsatz regulärer Ausdrücke nach verschiedenen Kriterien gefiltert werden sollen.

4.3 Automatisches Beantworten von Meldungen

Eine andere kommerziell z.Zt. sehr aktuelle Anwendung von Meldungsverarbeitung sind **E-Mail-Beantwortungs-Systeme**, also Systeme, welche die E-Mail von Kunden an Firmen nicht nur vorsortieren, sondern sogar mit einer entsprechenden Antwort versehen (und diese vor dem Versenden hoffentlich noch einem menschlichen Kontrolleur zur Freigabe vorlegen) sollen. Dabei wird versucht, für jede wichtige Frage die möglichen verschiedenen Formulierungen vorauszusehen, um jeweils die passende Antwort geben zu können.

Auch diese Systeme versuchen, mit regulären Ausdrücke auszukommen. Im Unterschied zu den reinen Filter- und Sortier-Systemen wird hier aber mit jedem regulären Ausdruck für eine spezifische Frage eine (vorgefertigte) Antwort verbunden. Da die möglichen Formen, die eine Frage annehmen kann, sehr verschieden sind, ergeben sich dabei ziemlich monströse reguläre Ausdrücke. Ein konkretes Beispiel für einen regulären Ausdruck, wie er in einem kommerziellen System[5] verwendet wird, kann als Illustration der Probleme dieses Ansatzes dienen (die verwendete Notation unterscheidet sich in einigen Punkten von der oben eingeführten (üblicheren) Repräsentation für reguläre Ausdrücke).

```
(((((((how/hwo/hou/hows/howz/how's/hows'))+(much/muhc))
/howmuch/hwomuch)&(money/cash/dough/bread/pesetas/peseta
/dollar/dollars/bucks/buck/quid/cents/euro/euros/dm/{$}
/usd/penny/quarter/dime/nickle))/((((how/hwo/hou/hows
/howz/how's/hows')/(what/wot/waht/wat/waddya/whut)/
(((what/wot/waht/wat/waddya/whut)+is)/whats/what's/wats
/wat's/wots/wot's/wahts/waht's/wotz))&(expensive
/expencive/much/muhc/cost/costs/(does+cost)/(do+cost)
/cots/expense/expence/expensivness/expensiveness/price
/(retail+price)/rp))/(((((how/hwo/hou/hows/howz/how's
/hows'))+(many/mayn/mani))/howmany/howmayn/((((how/hwo
/hou/hows/howz/how's/hows'))+(much/muhc))/howmuch/hwomuch))
&((to+pay+(for/fro))/(money/cash/dough/bread/pesetas
/peseta/dollar/dollars/bucks/buck/quid/cents/euro/euros/dm
/{$}/usd/penny/quarter/dime/nickle))))/(is+(affordable
/expensive/dear)/((cost/price/expense)+(high/low/great/small
```

5. der ‘‘Marc’’ Lingubot(TM) der Firma Kiwilogic, für die Firma Olympus entwickelt

```
/big/affordable))&%g?)/((can/may/might/possible/possibility)
&(buy/purchase/acquire/aquire)&(i/me))/((is/are)&
(purchaseable/acquireable/(for+sale)/(on+the+market)
/purchasable/acquirable)))))&((((eye/eyye/eyes/eie/e)
+(t/trek/treck/track/trak/treks/trecks/tracks/traks/thing
/thingamagic/thingamagik/thingi/thingo/things/thingumy
/thingummy/thinggy/thingy/gimmick/gimmic/gimmik/gimick
/gimic/gimik))/eything/eythingi/eythingo/eythings
/eythingumy/eythingummy/eythinggy/eythingy/eythingys
/eythingies/eyegimmick/eyegimmic/eyegimmik/eyegimick
/eyegimic/eyegimik/eyetrek/eyetreck/eyetrak/eyetrack/eytreck
/eitrek/eytrek/eytrack/eytrak/eyetreks/eyettrecks/eyetrecks
/eyetraks/eyetracks/eytrecks/eytrecs/eytracs/eytracks/eitreks
/eitrecs/unit/units/((the/them/this/these/those)&glasses/shades
/sunnies/sunglasses)/((it/one)&^eytrek)/{e-t})!{e.t.})
```

Man beachte, dass auch die Tippfehlerkorrektur in diesem Ausdruck erfasst wird.

Hierzu finden Sie hier eine ⇒ Lizentiatsarbeit: Björn Metzinger; Automatische E-Mail Beantwortung. Theoretische Grundlagen und Aspekte praktischer Anwendung; Lizentiatsarbeit; Sommer 2002.

4.4 ‘‘Chatterbots’’

Ebenfalls sehr aktuell sind schliesslich die sog. ‘‘*Chatterbots*’’ d.h. Systeme, welche z.B. mit Besuchern von Firmen-Webseiten in natürlicher Sprache konversieren sollen (siehe eine ⇒Übersicht).

Diese Systeme beruhen zum grössten Teil ebenfalls auf einfacher Mustererkennung (in Fragen erkannte Muster lösen die Ausgabe von ‘‘Konservenantworten’’ aus), was die Leistungsfähigkeit dieser Systeme in ganz prinzipieller Art beschränkt. Der vielleicht beste dieser ‘‘Chatterbots’’ ist ⇒ALICE , ein Open Source Projekt. Mehr dazu u.a. unter ⇒‘‘Chatterbots’’. Interessant an diesem System ist, dass eine spezifische Sprache für das Definieren der verwendeten Muster besteht, die AIML (Artificial Intelligence Markup Language). Aber es bleibt eben eine Sprache mit (offenbar nur) regulären Ausdrücken, und daher lässt sich auch ALICE leicht aufs Glatteis führen:

```
>Do you really understand human language?

ALICE: For real. Of course I understand it. Do you understand it?

>How do you do this?
```

```
ALICE: I am very well, how are you?

(> I meant how come you understand human languages? You are a
machine, after all.)

>You are a machine, after all. (Alice ignoriert den ersten Satz)

ALICE: Thanks and you are a delightful person too. Yes I am an
artificial brain.

>Do you know languages other than English?

ALICE: I know very little about it. What kind of it do you like?
```

Die ‘‘Urgrossmutter’’ aller Chatterbots ist ⇒ELIZA , die Karikatur eines Psychotherapeuten (die aber, dem Vernehmen nach, bei einigen Patienten durchaus Anklang fand). Das System existiert in der Zwischenzeit in einer Vielzahl von Varianten (z.B. eine Prolog-Version unter ⇒ ‘‘Eliza’’).

Ein typischer Dialog zeigt, was die Grenzen des Systems sind:

```
Hello. I am ELIZA. How can I help you?
> my mother hates me.

Tell me more about your family.
> Oh, well, I have two sisters. I hate them, too.

You say oh well you have two sisters you hate them too
> yes.

You seem quite positive.
> of course.

Lets discuss further why your mother hates you.
> She didn’t want me in the first place, I suppose.

Can you elaborate on that?

> Gosh, she just got married to my father for fear of becoming an old
spinster, and then she got pregnant. It’s as simple as that.

Who else in your family for fear of becoming an old spinster and then she
got pregnant its as simple as that?
```

```
> quit.

Goodbye. My secretary will send you a bill.
```

Während die Idee, einen Psychotherapeuten durch ein derartiges System zu ersetzen, als eher abartig erscheinen muss, sind Anwendungen in viel engeren (und weniger sensitiven) Bereichen nicht von vornherein ausgeschlossen. So kann man sich schon vorstellen, mittels regulärer Ausdrücke spezifische Fragen zu einem (eng begrenzten) Thema automatisch beantworten zu lassen. Ein derartiges System soll dabei die Antworten des Benutzers so auswerten, dass auch nicht-triviale Fragen einigermassen sinnvoll beantwortet werden können. Ds Ziel des Systems soll es also sein, einen Dialog wie etwa den folgenden zu erlauben:

```
User:       "My computer doesn’t work"
Lingubot:   "Do you see anything on your monitor?"
User:       "No"
Lingubot:   "Is the power indicator on your computer or on
             your monitor lit?"
User:       "The one on the computer is"
Lingubot:   "The one on your monitor isn’t?"
User:       "I don’t think so"
Lingubot:   "Please check the cable connecting the monitor to
             your computer.
             Is it correctly plugged in and are the safety
             screws fastened?"
```

4.5 ILAP

Im folgenden sollen die Möglichkeiten und Grenzen des Einsatzes von regulären Ausdrücken im beschriebenen Bereich anhand einer relativ einfachen Fragestellung vertieft werden. Die Aufgabe besteht darin, in einer Liste von Betreffzeilen einer Newslist folgende Arten von Zeilen zu identifizieren:

1. Aufrufe zur Einreichung von Konferenzbeiträgen (‘‘Calls for Papers’’), und zwar auch ‘‘Deadline extensions’’ für die Einreichung von Beiträgen etc.
2. Alle Meldungen zu Konferenzen (also auch Aufrufe zur *Teilnahme* an Konferenzen, Angaben zu *Daten* von Konferenzen etc.)
3. Job-Angebote (auch für PhD-Positionen etc.)

4. Spam (unerwünschte Werbemeldungen)

Sie sollten nun für jeden der vier Fälle jene Folge von regulären Ausdrücken finden, welche (in Sequenz auf die einzelnen Zeilen angewendet) alle, und nur, die gewünschten Zeilen identifizieren und damit herausfiltern. Ein Mustereintrag ist vorgegeben; er ist nicht unbedingt ideal und sollte modifiziert (oder entfernt) werden. Sie können (und sollen) auch mit den Suchoptionen experimentieren.

Das System meldet, welche Zeilen korrekterweise und welche fälschlicherweise gefunden worden sind, und welche korrekterweise und welche fälschlicherweise *nicht* gefunden worden sind (und den Prozentsatz solcher falsch positiver und falsch negativer Treffer). Durch systematisches Vergleichen dieser Angaben (und dem Inspzieren der Gesamtliste) sollten Sie ermitteln können, mit welchen regulären Ausdrücken Sie die korrekten Zeilen identifizieren können.

Natürlich werden Sie mit dieser Methode nur unvollkommene Ergebnisse erreichen. Erstens sagen Betreffzeilen oft nicht genu über den Inhalt einer Meldung aus, um eine klare Entscheidung über die Relevanz zuzulassen. In manchen Fällen wird Ihnen vielleicht Ihr Weltwissen sagen, wie eine Betreffziele kategorisiert werden muss, aber oft werden Sie auch als Mensch kapitulieren müssen (und der Computer umso mehr). Ganz besonders trifft dies bekanntlich auf Spam zu, wo Betreffzeilen oft bewusst irreführend formuliert werden. Zweitens ist das Werkzeug der regulären Ausdrücke eben recht grob - zu grob für die Aufgabe. Schliesslich kann nicht ausgeschlossen werden, dass einzelne der Meldung falsch vor-klassifiziert sind (denn die Korrektheit von Treffern muss natürlich gegen einen handgefertigten ‘‘Goldstandard’’ bestimmt werden, und beim Erstellen dieser Musterklassifikation können uns sehr wohl Fehler unterlaufen sein).

Grundlage des Experiments ist eine Liste von 2062 (echten, unmodifizierten) Betreffzeilen einer Newslist zum Thema der Künstlichen Intelligenz (`comp.ai.nat-lang`); Sie können sie von innerhalb der ILAP aus aufrufen (‘‘Totale Anzahl’’ anzeigen).

⇒**ILAP** (*Experiment*) Filtern von Betreffzeilen

5. Zusammenstellung: Alle interaktiven Elemente dieser Lerneinheit

⇒**QUIZ** Reguläre Ausdrücke: Eigenschaften

⇒**ILAP** *(Übung):* Pattern Matching (Zeichenketten)

⇒**ILAP** *(Übung):* Pattern Matching

⇒**ILAP** *(Übung):* Pattern Matching (Verankerung)

⇒**ILAP** *(Übung):* Pattern Matching (Optionalität, Gruppierung, Disjunktion)

⇒**ILAP** *(Übung):* Pattern Matching (Wiederholungen)

⇒**ILAP** *(Übung):* Pattern Matching (Numerische Wiederholungen)

⇒**ILAP** *(Übung):* Pattern Matching (Ersetzen mit regulären Ausdrücken)

⇒**QUIZ** Reguläre Ausdrücke: Basics

⇒**QUIZ** Reguläre Ausdrücke: Match

⇒**ILAP** *(Übung):* Praktische Einführung "Pattern Matching"

⇒**ILAP** (*Experiment):* Pattern Matching (Suchmustern allgemein)

⇒**ILAP** (*Experiment*) Datumserkennung

⇒**QUIZ** Datumserkennung mit Regulären Ausdrücken

⇒**ILAP** (*Experiment*) Filtern von Betreffzeilen

6. Zitierte Literatur

7. Inhaltsverzeichnis