Wiederholung

Ein *deterministischer endlicher Automat* (DEA) über einem Alphabet A besteht aus:

- einer *endlichen* Menge von Zuständen Q,
- einem Anfangszustand $q^0 \in Q$,
- einer Menge von Endzuständen $Q^E \subseteq Q$,
- einer Übergangsfunktion $\delta : Q \times A \to Q$.

Wiederholung

Ein DEA beginnt im Anfangszustand q^0.

Er liest die Zeichen des Wortes sequentiell von vorn nach hinten.

Wenn sich der Automat in einem Zustand q befindet und ein Zeichen x liest, geht er in den Zustand $\delta(q, x)$ über.

Wenn er sich am Wortende in einem Zustand aus Q^E befindet, *akzeptiert* er das Wort.

Die Menge aller Wörter, die ein Automat akzeptiert,
ist die *von diesem Automaten akzeptierte Sprache*.

Wiederholung

Nichtdeterministischer endlicher Automat (NEA):

- Verallgemeinerung deterministischer endlicher Automaten:
- Statt Übergangsfunktion: Übergangsrelation,
  d. h.: für einen Zustand q und ein Zeichen x kann es einen, mehrere, oder auch keinen Folgezustand geben.
- Zusätzlich möglich: ε-Übergänge,
  d. h.: der Automat wechselt von einem Zustand in einen anderen, ohne ein Zeichen zu lesen.

Wiederholung

Definition: Ein NEA akzeptiert ein Wort, wenn es mindestens eine Möglichkeit gibt, das Wort *vollständig* zu lesen und dabei vom Anfangs- in einen Endzustand zu gelangen.

Satz: Für jede Sprache, die von einem NEA akzeptiert wird, existiert auch ein DEA, der sie akzeptiert.
(Die Umkehrung ist trivial, denn jeder DEA ist ein NEA.)

Wiederholung

Eine Sprache L heißt regulär, wenn es einen DEA (oder NEA) gibt, der L akzeptiert.

Beispiele für nicht-reguläre Sprachen:

- $\{\, a^n b^n \mid n \geq 0 \,\}$.
- Menge aller „korrekt geklammerten" Wörter über dem Alphabet $\{(,)\}$:

Grund: Endliche Automaten können nicht beliebig weit zählen.

Wiederholung

Reguläre Sprachen sind unter vielen Operationen abgeschlossen:

Wenn L eine reguläre Sprache ist, dann sind ebenfalls regulär:

- das Komplement $A^* \setminus L$,
- der Abschluß L^*
- und der nichtleere Abschluß L^+.

Wenn L_1 und L_2 reguläre Sprachen sind, dann sind ebenfalls regulär:

- der Durchschnitt $L_1 \cap L_2$,
- die Vereinigung $L_1 \cup L_2$,
- und die Konkatenation $L_1 \cdot L_2$.

Reguläre Ausdrücke

Sei A ein Alphabet.

Wir bezeichnen $\emptyset$, $\{\varepsilon\}$ und $\{x\}$ (für jedes $x \in A$) als Grundsprachen.

Wir bezeichnen die Operationen Vereinigung ($L_1 \cup L_2$), Konkatenation ($L_1 \cdot L_2$) und Abschluß (L^*) als reguläre Operationen.

Satz: Eine Sprache ist genau dann regulär, wenn sie sich aus den Grundsprachen mittels der regulären Operationen aufbauen läßt.

Beweis:
($\Leftarrow$) Die Grundsprachen sind regulär, die regulären Operationen erhalten die Regularität.

($\Rightarrow$) Sehr technisch.

Reguläre Ausdrücke

Reguläre Ausdrücke beschreiben den Aufbau einer Sprache aus Grundsprachen mittels regulärer Operationen:

Das Symbol $\emptyset$ ist ein regulärer Ausdruck, der die leere Menge $\emptyset$ beschreibt.

Das Symbol ε ist ein regulärer Ausdruck, der die Sprache $\{\varepsilon\}$ beschreibt, die nur aus dem leeren Wort besteht.

Wenn x ein Zeichen aus A ist, dann ist x ein regulärer Ausdruck, der die Sprache $\{x\}$ beschreibt, die nur aus dem Wort x besteht.

Reguläre Ausdrücke

Wenn r_1 und r_2 reguläre Ausdrücke sind, die die Sprachen L_1 und L_2 beschreiben,
dann ist $r_1 r_2$ ein regulärer Ausdruck, der die Sprache $L_1 \cdot L_2$ beschreibt,
und $r_1 | r_2$ ist ein regulärer Ausdruck, der die Sprache $L_1 \cup L_2$ beschreibt.

Wenn r ein regulärer Ausdruck ist, der die Sprache L beschreibt,
dann ist r^* ein regulärer Ausdruck, der die Sprache L^* beschreibt.

Zusätzlich: Klammern nach Bedarf.

Reguläre Ausdrücke

Beispiele:

Der reguläre Ausdruck $a(b|c)$ beschreibt die Sprache $\{ab, ac\}$.
Auch der reguläre Ausdruck $ab|ac$ beschreibt diese Sprache.

Der reguläre Ausdruck ab^* beschreibt die Sprache $\{a, ab, abb, abbb, \ldots\}$.

Der reguläre Ausdruck $(ab)^*$ beschreibt die Sprache $\{\varepsilon, ab, abab, ababab, \ldots\}$.

Reguläre Ausdrücke

Die folgenden Eigenschaften sind äquivalent:

- Die Sprache L ist regulär.
- Es existiert ein DEA D, der L akzeptiert.
- Es existiert ein NEA N, der L akzeptiert.
- L läßt sich aus den Grundsprachen mittels der regulären Operationen aufbauen.
- Es existiert ein regulärer Ausdruck r, der L beschreibt.

Reguläre Ausdrücke

Die verschiedenen Darstellungen von L lassen sich automatisch mittels eines Programms ineinander umrechnen.

In der Praxis:

- Benutzer spezifiziert L mittels regulärem Ausdruck r;
- Programm konvertiert r in DEA oder NEA und untersucht einen gegebenen Text mit diesem Automaten.

Reguläre Ausdrücke in der Praxis

Wo findet man reguläre Ausdrücke?

- Programmiersprachen:
  - eingebaut: Perl, Python, AWK, sed.
  - als zusätzliches Package: Java (z. B.: org.apache.regexp, org.apache.oro.text.regex, gnu.regexp)
- Suchprogramme: z. B. grep.
- Editoren: z. B. alle emacs- und vi-artigen Editoren.
- Pager: z. B. less, more.
- Scannergeneratoren: z. B. lex, flex, JLex.

Reguläre Ausdrücke in der Praxis

Reguläre Ausdrücke, wie sie in praktischen Implementierungen verwendet werden, unterscheiden sich in einigen Punkten von der theoretischen Definition.

Reguläre Ausdrücke in der Praxis

Problem 1:

In der Theorie:

- Sprache besteht aus Wörtern über einem beliebigen endlichen Alphabet A;
- reguläre Ausdrücke verwenden zusätzliche Zeichen $\emptyset$, ε, $|$, *, $(\ldots)$, die nicht in A enthalten sind (sogenannte Meta-Zeichen).

In der Praxis:

- A = ASCII oder ISO-8859-n oder UNICODE;
- zur Darstellung regulärer Ausdrücke stehen keine „neuen", „bisher ungenutzen" Zeichen zur Verfügung.

Reguläre Ausdrücke in der Praxis

Ausweg:

- Bestimmte Zeichen aus A werden sowohl als „normale" als auch als Meta-Zeichen verwendet;
- zur Unterscheidung wird in einem der beiden Fälle ein „Fluchtsymbol" davorgesetzt (gewöhnlich „\").

Beispiel:

- `a*` bezeichnet die Sprache $\{\varepsilon, \mathtt{a}, \mathtt{aa}, \mathtt{aaa}, \ldots\}$.
- `a\*` bezeichnet die Sprache $\{\mathtt{a*}\}$.

Reguläre Ausdrücke in der Praxis

Problem 2:

In der Theorie:

Die Sprache, die aus den Kleinbuchstaben von a bis z besteht, wird durch den regulären Ausdruck $(a \mid b \mid c \mid d \mid e \mid f \mid g \mid h \mid i \mid j \mid k \mid l \mid m \mid n \mid o \mid p \mid q \mid r \mid s \mid t \mid u \mid v \mid w \mid x \mid y \mid z)$ beschrieben.

In der Praxis:

Abkürzungen sind unbedingt notwendig.

Reguläre Ausdrücke in der Praxis

Abkürzungen:

`.`	ein beliebiges Zeichen
`[aeiou]`	eines der Zeichen `a`, `e`, `i`, `o`, `u`.
`[a-dmx-z]`	eines der Zeichen `a`, `b`, `c`, `d`, `m`, `x`, `y`, `z`.
`[^aeiou]`	ein beliebiges Zeichen außer `a`, `e`, `i`, `o`, `u`.
`[^a-dmx-z]`	ein beliebiges Zeichen außer `a`, `b`, `c`, `d`, `m`, `x`, `y`, `z`.

Reguläre Ausdrücke in der Praxis

Problem 3:

In der Theorie:

Gegeben: ein Wort w, ein regulärer Ausdruck r.
Frage: Ist w in der durch r definierten Sprache enthalten?
(„Paßt der reguläre Ausdruck r auf w?“
„*Matcht* der reguläre Ausdruck r das Wort w?“)

In der Praxis:

Häufig lautet die Frage: sind irgendwelche *Teilwörter von* w in der durch r definierten Sprache enthalten?
(Und wenn ja: welche Teilwörter?)

Reguläre Ausdrücke in der Praxis

Konvention:

Normalerweise wird untersucht, ob ein regulärer Ausdruck auf *ein Teilwort* des untersuchten Wortes paßt.
Für den Fall, daß man das nicht will, gibt es zwei Sonderzeichen:

`^` vor dem regulären Ausdruck:
paßt nur am Anfang des Wortes (d. h. meist: der Zeile)

`$` nach dem regulären Ausdruck:
paßt nur am Ende des Wortes (d. h. meist: der Zeile)

Reguläre Ausdrücke in der Praxis

Beispiel:

`ab` paßt auf `ab`, `abccc`, `cccab`, `cccabccc`.

`^ab` paßt auf `ab`, `abccc`.

`ab$` paßt auf `ab`, `cccab`.

`^ab$` paßt auf `ab`.

Zusammengefaßt

Die Implementierungen regulärer Ausdrücke in verschiedenen Programmen oder Programmiersprachen sind leider nicht einheitlich. Wir verwenden im folgenden die Syntax der Programmiersprache Perl. Andere Implementierungen weichen davon teilweise ab.

In Perl gilt folgende Konvention:
Alphanumerische Zeichen ohne \ stehen für sich selbst.
Nicht-alphanumerische Zeichen mit \ stehen für sich selbst.
Die anderen Kombinationen haben möglicherweise eine Meta-Bedeutung.
(weiteres: man perlre)

Zusammengefaßt

`.` ein beliebiges Zeichen

`[0-9A-F]` Zeichenklasse

`[^0-9A-F]` negierte Zeichenklasse

`|` Vereinigung/Alternative

`*` Abschluß (vorstehender Ausdruck beliebig oft)

`+` nichtleerer Abschluß
(vorstehender Ausdruck mindestens einmal, `A+` = `AA*`)

`?` vorstehender Ausdruck ein- oder keinmal (`A?` = `(A|)`)

`^` Zeilenanfang

`$` Zeilenende

`(...)` Klammern von Ausdrücken

Zusammengefaßt

`\t` Tab

`\n` Newline (LF)

`\r` Return (CR)

`\w` ein „Wort“-Zeichen (alphanumerisches Zeichen oder _)

`\s` ein Whitespace-Zeichen (Leerzeichen, Tab, LF, CR)

`\d` eine Ziffer

`\b` eine „Wort“-Grenze (Anfang oder Ende einer Folge von `\w`-Zeichen)

Zusammengefaßt

Vorsicht:

`ac*` *paßt* am Anfang von `abc`.

`a[^b]` bedeutet *nicht* „a nicht gefolgt von `b`“.

Die Schreibweise `[^...]` dient nur zur Negation von *Zeichenklassen*; man kann damit nicht beliebige reguläre Ausdrücke komplementiern.

Beispiel

Beispiel: Analyse eines WWW-Logfiles.

Die Einträge im Logfile eines WWW-Servers sehen wie folgt aus (eine Zeile!):

```
isis.informatik.uni-kl.de - - [20/Mar/1998:02:21:31 +0100] \
"GET /~uwe/paper/AAECC94-bibl.html HTTP/1.0" 200 2201
```

Um herauszufinden, welche Postscript-Dateien (`.ps`) wann und von wo angefordert wurden, kann man den folgenden regulären Ausdruck benutzen:

```
".*\.ps␣
```

Die vorletzte Zahl in der Zeile ist der Return-Code des Zugriffsversuchs. Return-Codes, die mit `4` anfangen, signalisieren Fehler. Man findet fehlgeschlagene Zugriffsversuche also mit dem regulären Ausdruck

```
"␣4
```


Wie paßt ein regulärer Ausdruck?

Wie paßt ein regulärer Ausdruck auf einen String?
(„Wie matcht der reguläre Ausdruck?“)

Beispiel:

Regulärer Ausdruck: `ab*`
String: `XXabYYabbZZ`
Der reguläre Ausdruck paßt auf das 3. Zeichen,
das 3. und 4. Zeichen,
das 7. Zeichen,
das 7. und 8. Zeichen,
das 7., 8. und 9. Zeichen.

Wenn man einem Editor den Befehl gibt, „`ab*`“ durch ein Leerzeichen „␣“ zu ersetzen, welche Zeichen werden ersetzt?

Wie paßt ein regulärer Ausdruck?

Übliche Konvention:

Unter allen möglichen Matches, nimm die, die am frühesten anfangen.

Nimm davon diejenigen Matches, bei denen der erste Teilausdruck am weitesten reicht. Nimm davon diejenigen Matches, bei denen der zweite Teilausdruck am weitesten reicht. ... – bis nur ein Match übrig bleibt.

Beispiel:

Regulärer Ausdruck: `[abc]*b[abc]*c`
String: `abcabaacaacaabaa`

Suchen und Ersetzen

Zugriff auf Teilausdrücke:

Programme, die eine Suche-und-Ersetze-Funktion mit regulären Ausdrücken zur Verfügung stellen, erlauben es üblicherweise, in dem Ersetzungsstring auf Teile des gematchten Ausgangsstrings zuzugreifen.

Wenn im Ersetzungsstring $1, $2, ... auftreten, dann wird an dieser Stelle derjenige Teilstring eingesetzt, der vom ersten/zweiten/... *geklammerten* Teilausdruck gematcht wurde.

Suchen und Ersetzen

Ersetzungsanweisung in Perl: `s/.../.../;`

Ersetze einen String, der aus einer Folge von Nicht-Leerzeichen, einem Leerzeichen und einer weiteren Folge von Nicht-Leerzeichen besteht, durch die zweite Folge von Nicht-Leerzeichen, ein Komma und die erste Folge von Nicht-Leerzeichen:

```
perl -pe 's/([^ ]*) ([^ ]*)/$2,$1/;'
```


Beispiel

Beispiel: Analyse eines WWW-Logfiles.

```
isis.informatik.uni-kl.de - - [20/Mar/1998:02:21:31 +0100] \
"GET /~uwe/paper/AAECC94-bibl.html HTTP/1.0" 200 2201
```

Wie kann man einfach ein WWW-Logfile nach der Toplevel-Domain (`.de`, `.fr`, `.com`, ...) sortieren?

Unter Unix steht ein Sortierprogramm (`sort`) zur Verfügung; aber der Teilstring, nach dem sortiert werden soll, ist für `sort` zu versteckt.

Idee: Kopiere die Toplevel-Domain an den Anfang jeder Zeile:

```
de isis.informatik.uni-kl.de - - [20/Mar/1998:02:21:31 +0100] \
"GET /~uwe/paper/AAECC94-bibl.html HTTP/1.0" 200 2201
```

Die Lösung:

```
perl -pe 's/([^ ]*)\.([^ ]*)/$2 $1.$2/;' weblog | sort
```


Beispiel

Beispiel: Namensliste konvertieren (Karl Egon Meier $\rightsquigarrow$ Meier, Karl Egon).

```
s/(.*) (.*)/$2, $1/;
```

Problem: mehrteilige Nachnamen (von Goethe, Mac Donald, Di Caprio)

```
s/(.*) (.*)/$2, $1/;
s/(.*) ([a-z]+|Ma?c|D[aei]|L[ae])$/$2 $1/;
```

Tip: Daten vorher „desinfizieren“:

```
s/\t/ /g; ## Tab -> Leerzeichen
          ##   (g = global, d.h.: alle Vorkommen ersetzen)
s/ +/ /g; ## beliebige Folge von Leerzeichen
          ##   -> ein Leerzeichen
s/^ //;   ## Leerzeichen am Anfang löschen
s/ $//;   ## Leerzeichen am Ende löschen
```

Beispiel

Beispiel: Erstes und zweites Argument einer Methode f(x,y,z) vertauschen.

Problem: Um Aufrufe wie `f(g(x,h(y)),z,w)` richtig umformen zu können, müßte man Klammern zählen. Das geht mit regulären Ausdrücken nicht.

Wir können aber immerhin vorher überprüfen, ob solche kritischen Fälle in der Datei auftauchen:

Taucht eine öffnende Klammer vor dem ersten Komma nach „`f(`“ auf?

```
\bf\s*\([^(,]*\(
```

Taucht eine öffnende Klammer zwischen dem ersten und dem zweiten Komma auf?

```
\bf\s*\([^,]*,[^(,]*\(
```

Wenn nein, dann funktioniert folgende Ersetzung:

```
s/\bf\s*\(([^),]*),([^),]*)/f($2,$1/g;
```