Reguläre Ausdrücke

Reguläre Ausdrücke spielen in der computerunterstützten Textverarbeitung allgemein und in der Sprachtechnologie im Besonderen eine herausragende Rolle.

Die komplexe Notation ist zwar abschreckend, wenn man sie aber einmal durchschaut hat, sind reguläre Ausdrücke eigentlich ganz einfach und vielseitig einsetzbar.

Reguläre Ausdrücke sind nach festen Regeln gebildetet Zeichenfolgen, die als Muster (engl. *pattern*) für die Suche und Bearbeitung von Textfragmenten in Texten dienen. Die grundlegende Operation ist der Mustervergleich (engl. *pattern matching*).

Vom Konzept her kommen solche Ausdrücke – wenn auch in anderer Notation – in der Linguistik häufig zum Einsatz. Diesen Umstand wollen wir als Einstieg benutzen, um auf einer uns bekannten Basis ein paar Grundlagen zu klären

Einstieg: Reguläre Ausdrücke in der Syntax

Bei dem folgenden Beispiel ist zu berücksichtigen, dass die Grundeinheit, um die es hier geht, das Wort bzw. die Wortform ist. Wie wir später sehen werden, ist das im Bereich der Spachtechnologie anders.

Gegeben seien die folgenden Daten:

1. a very clever student
2. a rather clever student
3. a clever student
4. the lazy teacher
5. the cat

Diese Daten können wir wie in der nachstehenden Graphik repräsentieren:

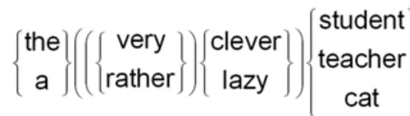


Abb. 1.

Bei **Abb. 1** handelt es sich bereits um einen regulären Ausdruck (fortan *RA*), der die Abfolgemuster der nebenstehenden Daten erfasst. Per Konvention werden in der Syntax fakultative Elemente durch runde, Alternativen durch Schweifklammern repräsentiert.

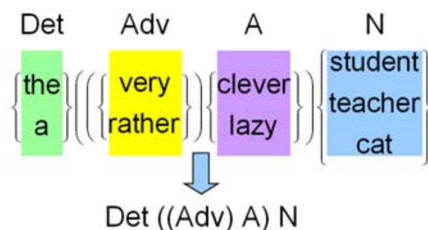


Abb. 2

Wenn wir den RA in **Abb.1** alsGrundlage nehmen derart, dass wir die konkreten Wortformen durch ihre jeweiligen Kategorial-symbole ersetzen, landen wir bei einem RA, der uns als Regelrumpf aus PS-Regeln für Nominalphrasen wohlbekannt ist, wie **Abb. 2** zeigt.

Derartige RA in der Syntax eignen sich gut dafür, im Rahmen von Übergangsnetzwerken dargestellt und implementiert zu werden. Dazu ein Beispiel:

Daten:	**Lexkats**:
John cried	Name Verb_i
The boy jumped	Det Nomen Verb_i
John admired Mary	Name Verb_t Name
The boy kicked the ball	Det Nomen Verb_t Name
Mary read a book	Name Verb_t Det Nomen
The boy hit John	Det Nomen Verb_t Name

RA:

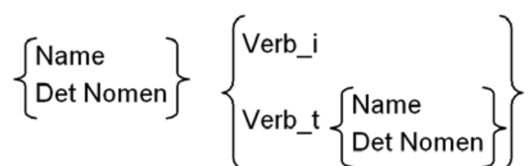


Tabelle 1

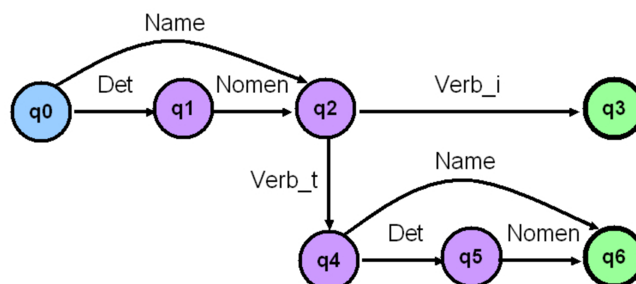


Abb. 3: der RA ausTabelle 1 als einfaches Übergangsnetzwerk[1]

[1] Das Übergangsnetzwerk hätte auch eine andere Form haben können, beispielsweise hätte q4 über q5 auch mit q3 verbunden sein können, oder es hätten Teilnetze wie das NP-Netz ausgelagert werden können.

Reguläre Ausdrücke im Kontext der Textverarbeitung und Sprachtechnologie

Einleitung

Wir beginnen mit einem einfach Beispiel. Gesetzt den Fall, wir wollten in einem Text nach der folgenden Zeichenfolge suchen: 1111123456. Wir können das tun, indem wir ein Muster vorgeben, das identisch ist mit genau dieser Zeichenfolge: 1111123456. Per Konvention aber können wir das auch anders machen, nämlich dadurch, dass wir das Muster wie folgt notieren: 1{5}2345. Das '{5}' bezieht sich auf das unmittelbar vorhergehende Zeichen und bedeutet, dass dieses genau fünf Mal vorzukommen hat (mehr dazu s.u.). Wir haben also hier die Wahl zwischen Muster_1 und Muster_2:

Zeichenfolge: 111112345

Muster_1: 111112345

Muster_2: 1{5}2345

Die Erkenntnis behalten wir erst einmal im Hinterkopf und betrachten das folgende Beispiel.

aaabccc a{3}bc{3}

aaadccc a{3}dc{3}

Wir haben zwei Zeichenfolgen und zwei dazugehörige RA. Interessant wird Sache mit den RA aber erst dann, wenn wir aus diesen beiden Mustern ein Muster erzeugen wollen, das für **beide** Zeichenketten anwendbar ist.

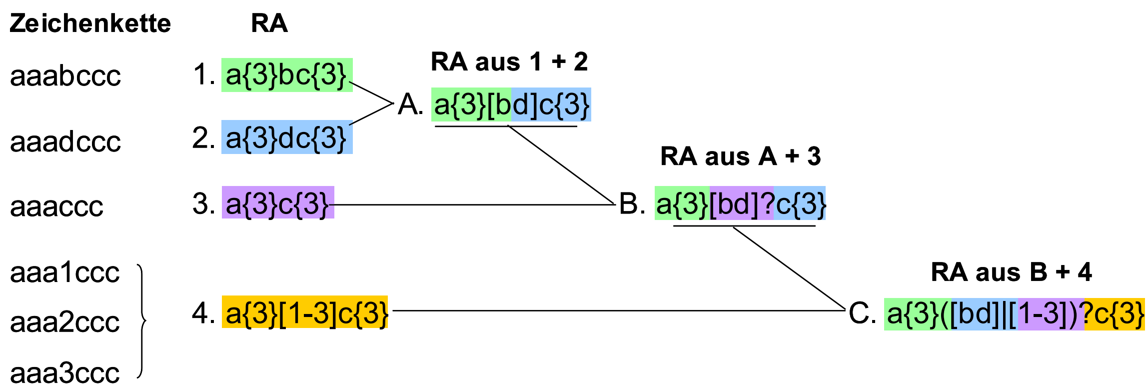
Wenn wir die fraglichen Zeichenfolgen analysieren, stellen wir fest, dass die Elemente b oder d Alternativen zueinander darstellen. Eine Möglichkeit, das in Form eines RA auszudrücken, besteht darin, diese Elemente in eckige Klammern zu setzen. Ein Muster wie z.B. 1[23]45 wäre somit auf die Zeichenfolgen 1245 und 1345 anwendbar.[2] Damit erhalten wir die folgende Situation:

Zeichenkette:	RA
aaabccc	1. a{3}bc{3}
aaadccd	2. a{3}dc{3}

RA aus1 & 2

a{3}[bd]c{3}

Nähmen wir auch noch die Zeichenfolgen aaaccc, aaa1ccc, aaa2ccc und aaa3ccc mit entsprechenden Mustern hinzu, können wir die Tabelle wie folgt renotieren, wobei wir die Konventionen für Fakultativität und Alternativen (die weiter unten genau erklärt werden) berücksichtigen. Die Farben sollen dabei nicht hübsch aussehen, sondern anzeigen, was sich woraus zusammensetzt (ok – sie sollen auch hübsch aussehen...).



Mit RA C sind also alle Zeichenfolgen in der Tabelle beschrieben, und genau das ist Grundprinzip und die Grundfunktion regulärer Ausdrücke: diese sind so fomuliert, dass sie maximal generalisiert Muster in Zeichenfolgen identifizieren.

Die Struktur regulärer Ausdrücke im Bereich Textverarbeitung und Sprachtechnologie

Im Grunde gilt hier dasselbe Prinzip, wie in der Syntax: RAs sind nach festen Regeln gebildete Ausdrücke, die als Muster für Zeichenfolgen dienen. Was anders ist, als in der Syntax, sind die Grundeinheiten um die es geht: nicht Wörter bzw. Wortformen, sondern Zeichen. 'Zeichen' ist hier nicht im semiotischen Sinn zu verstehen, sondern meint konkret das, was im Englischen mit *character* bezeichnet ist, also Buchstaben, Ziffern, Interpunktionszeichen usw. Da also diese Grundeinheiten verschieden sind von denen der Syntax, sind logischerweise auch die Klassen, mit denen wir hier arbeiten, andere. Ferner sind die verwendeten Konventionen für die Darstellung von Wiederholung, Fakultativität und Alternationen andere.

Grundeinheiten und Klassen

Anders als in der Syntax sind die Grundeinheiten keinen Wörter, sondern Zeichen. Eine für die lexikalische Analyse durchgeführte Tokenisierung der Kette 'The cat slept.' würde uns als Ergebnis die Tokenliste ['The',

[2] Eine genaue Erklärung dieser Konvention erfolgt weiter unten

'cat', 'slept'] zurückliefern. Was wir aber jetzt wollen, sind die einzelnen Zeichen als Token, also ['T','h','e',' ', 'c','a','t',' ','s','l','e','p','t' '.']. Da wir es also mit gänzlich anderen Grundeinheiten zu tun haben, sind auch die Klassen, mit denen wir operieren, andere, z.B.

- Großbuchstaben
- Kleinbuchstaben
- Ziffern
- Interpunktionszeichen
- Separatoren

Um diese Zeichenklassen zu definieren bzw. sich auf deren Elemente zu beziehen, gilt es, bestimmte Konventionen zu berücksichtigen. Zunächst einmal stellen wir fest, dass wir es in allen Fällen mit geschlossenen Klassen zu tun haben. Bei diesen bietet sich z.B. eine **extensionale Definition** an, die durch Aufzählung aller Elemente, die die Klasse konstituieren, erzielt wird. Beispiele:

- Klasse 1: Alle Großbuchstaben: [ABCDEFGHIJKLMNOPQRSTUVWXYZ]
- Klasse 2: Alle Großbuchstaben von P...Y: [PQRSTUVWXY]
- Klasse 3: Alle Ziffern: [0123456789]
- Klasse 4: Alle Ziffern von 3...9: [3456789]
- Klasse 5: Alle Ziffern von 2...6 und alle Großbuchstaben von C...F: [23456CDEF]

Das ist aber auf Dauer zu umständlich und würde zu komplett unleserlichen RA führen. Da wir es im Falle der Buchstaben und Ziffern mit geordneten Mengen[3] zu tun haben, verwenden wir das Verfahren der **Bereichsnotation** für die Definition der Klasse. Beispiele:

- Klasse 1: Alle Großbuchstaben: [A–Z]
- Klasse 2: Alle Großbuchstaben von P...Y: [P–Y]
- Klasse 3: Alle Ziffern: [0–9]
- Klasse 4: Alle Ziffern von 3...9: [3–9]
- Klasse 5: Alle Ziffern von 2...6 und alle Großbuchstaben von C...F: [2–6C–F]

Im praktischen Gebrauch kann sich aber auch das Schema für die Bereichsnotation als umständlich erweisen. Deshalb gibt es für bestimmte Klassen vordefinierte **Bezeichnersymbole**, u.a. 'd' für Digit, 's' für *Whitespace* und 'w' für Wortzeichen. Um diese Bezeichner nicht mit den tatsächlichen Buchstaben d, s und w zu verwechseln, werden sie durch ein vorangestelltes Escape-Zeichen (\) gekennzeichnet. Es gilt also

\d Klassenbezeichner für Ziffern

\s Klassenbezeichner für *Whitespace*

\w Klassenbezeichner für Wortzeichen

Somit haben wir für die Klasse aller Ziffern drei verschiedene Definitionsmöglichkeiten:

[0123456789] extensionale Definition durch Aufzählung

[0–9] Bereichsnotation

\d Bezeichnersymbol

Drei Möglichkeiten gibt es auch für die Klasse der Wortzeichen, der alle Buchstaben, Ziffern und der Unterstrich angehören:

[ABCDEFGHIJKLM
NOPQRSTUVWXY
Zabcdefghijklmnop
qrstuvwxyz0123456
789_] } extensionale Definition durch Aufzählung

[A–Za–z0–9_] Bereichsnotation

\w Bezeichnersymbol

Bei der Klasse *Whitespace* allerdings ist keine Bereichsnotation möglich: diese ist ja nur dann anwendbar, wenn es sich um eine geordnete Menge von Zeichen handelt, und das ist bei *Whitespace* nicht der Fall:

[\t\n] extensionale Definition durch Aufzählung[4]

\s Bezeichnersymbol

Interessant ist in diesem Zusammenhang auch der Umstand, dass wir **Komplementärklassen** definieren können. In der Mengenlehre ist die Komplementärmenge A' einer Menge A die Menge aller Elemente, die nicht zu A gehören. Um auf die Klasse der Ziffern zurückzukommen: die Komplementärklasse der Ziffern

[3] 'Geordnet' deshalb, weil die zugrundeliegende Kodierung geordnet ist. Im Ascii-Code z.B. haben die Buchstaben ABCDE die Codenummern 65-66-67-68-69. Die Ziffern '0'–'9' haben die Codewerte 48–58 usw.

[4] Das Spatium am Ende ist absichtlich: das Leerzeichen gehört neben und Tab und Zeilenwechsel natürlich zur Klasse *Whitespace*

wäre, umgangssprachlich ausgedrückt, 'alles, was keine Ziffer ist'. Im Rahmen regulärer Ausdrücke wird bei extensionaler Definition und Bereichsnotation die Komplementärklasse durch ein Zirkumflex ausgdrückt, das der Klasse, um die es geht, innerhalb der eckigen Klammer vorangestellt ist:

Klasse	Komplementärklasse
[0–9]: alle Ziffern von 0 - 9	[^0–9]: alles, was keine Ziffer von 0 - 9 ist
[ABCDE]: Alle Großbuchstaben von A - E	[^ABCDE]: alles, was kein Großbuchstabe von A - E ist
[s]: der Kleinbuchstabe 's'	[^s]: alles, was kein Kleinbuchstabe 's' ist (≈ kein 's')

Bei den Bezeichnersymbolen wird die Komplementärklasse allerdings nicht durch das Zirkumflex dargestellt, sondern durch den jeweiligen Großbuchstaben:

Klasse	Komplementärklasse
\d: alle Ziffern von 0 - 9	\D: alles, was keine Ziffer von 0 - 9 ist
\s: Klasse der *Whitepaces*	\S: alles, was kein *Whitespace* ist
\w: Klasse der Wortzeichen	\W: alles aus der Grundmenge zulässiger Zeichen, was kein Wortzeichen ist

Klassen von Grundeinheiten — wahllose Beispiele:

[0-9aw-y] : [0123456789awxyz]
[ABd-g] : [ABdefg]
[sab\s] : [sab\t\n]
\s : [\t\n]
\S : alles außer [\t\n]

Der Einsatz von Klassen in RA

Dieser Punkt muss, um Missverständnisse zu vermeiden, geklärt werden, bevor wir uns im weiteren Text mit RA-spezifischen Konventionen für Fakultativität, Alternativen und Wiederholungen beschäftigen. Nehmen wir gleich ein konkretes Beispiel, den RA **ab[0-4]cd**. Hier könnte man mit der Bereichsnotation (und auch der extensionalen Definition) der Klasse [0-4] ins Schleudern kommen, wenn diese im Rahmen eines RA eingesetzt wird, und vermuten, dieser RA sei ein Muster für die Zeichenkette ab01234cd. Das ist aber **falsch**: der RA ist ein Muster für eine Kette bestehend aus **5** Zeichen, nämlich

1. Zeichen	2. Zeichen	3. Zeichen	4. Zeichen	5. Zeichen
Der Buchstabe 'a'	Der Buchstabe 'b'	Ein Element der Klasse [01234]	Der Buchstabe 'c'	Der Buchstabe 'd'

Der RA bildet mithin die folgenden Zeichenketten ab: ab0cd, ab1cd, ab2cd, ab3cd und ab4cd. Er ist **kein** Muster für die Zeichenkette ab01234cd.

Wiederholungen, Fakultativität und Alternativen in RA

Wiederholungen werden in ihrer allgemeinsten Form durch einen dem Zeichen nachgestellten Ausdruck der Form {min,max} dargestellt. 'min' gibt die Untergrenze an, 'max' die Obergrenze der Wiederholungen:

- X{4,7} bedeutet *mindestens vier aber höchstens sieben X*
- \d{2,4} bedeutet *mindestens zwei aber höchstens vier Ziffern*

Wenn min und max identisch sind, werden sie nicht einzeln ausgewiesen. Ein Ausdruck wie \d{3} ist eine Kurzform für \d{3,3}: mindestens drei und höchstens drei Ziffern, also genau drei Ziffern.

- \w{5} bedeutet *genau 5 Wortzeichen*

Ein Ausdruck wie [ab]{2} bedeutet *genau 2 Elemente aus der Klasse [ab]* und identifiziert die Zeichenketten *aa*, *ab*, *ba* und *bb*.

Ein Ausdruck wie a{0,}, bei dem die Untergrenze mit 'Null' angegeben und die Obergrenze offen gelassen wird, bedeutet "0-mal a oder öfter". Für diesen Inhalt wird auch der Kleene-Stern verwendet:

- [A-Z]* bedeutet *kein Element aus A-Z oder unbegrenzt viele*

Ein Ausdruck wie a{1,}, bei dem die Untergrenze 1 ist und die Obergrenze offen, bedeutet "mindestens 1-mal a oder öfter". Für diesen Inhalt wird auch das Kleene-Plus verwendet:

- [A-Z]+ bedeutet *mindestens ein Element aus A-Z oder unbegrenzt viele*

Ein Ausdruck wie a{0,1}, bei dem die Untergrenze 0 ist und die Obergrenze 1, bedeutet "0-mal a oder einmal a". Für diesen Inhalt wird auch das Fragezeichen verwendet, und so wird in RA **Fakultativität** ausgedrückt:

- [A-Z]? bedeutet *kein Element aus A-Z oder ein Element aus A-Z*

Eine Möglichkeit dafür, **Alternativen** in einem RA auszudrücken, haben wir weiter oben kennengelernt, nämlich durch den Einsatz entsprechend definierter Klassen. Der RA ab[cd] ist wie folgt zu lesen:

- ab[cd] bedeutet *erst a, dann b, dann ein Element aus der Klasse [cd]*

Den letzten Teil dieser Aussage könnte man genau so gut mit ***dann entweder c oder d*** paraphrasieren: eine klassische Alternative.

Etwas anders sieht es im folgenden Beispiel aus. Grundlage sind die nachstehenden Zeichenketten, insbesondere die darin fettgedruckten Ketten: ab**xyz**cd und ab**123**cd. Um einen RA für beide Ketten zu formulieren, müssen wir ausdrücken, dass die Teilketten xyz und 123 im gleichen Kontext füreinander austauschbar sind. Dabei können wir aber keine Klassen einsetzen. Stattdessen gehen wir so vor, dass wir zwischen die beiden Alternanten eine senkrechte Linie als *oder*-Operator setzen: xyz|123. Dies ist als *entweder xyz oder 123* zu lesen. Im RA muss dieses Disjunktion allerdings dann durch runde Klammern eingeschlossen werden: ab(xyz|123) cd. Dieser RA ist wie folgt zu lesen:

- ab(xyz|123)cd bedeutet *erst a, dann b,* ***dann entweder xyz oder 123****, dann c, dann d*

Hier sieht man eine Funktion der runden Klammern: sie sind dabei behilflich, auszudrücken, was sich worauf bezieht. Vergleichen Sie die RA abc{3}de vs a(bc){3}de: im ersten Fall wäre die abgebildete Zeichenkette abcccde, im zweiten dagegen abcbcbcde.

Rückbezüge in RA

Eine weitere Aufgabe der runden Klammern besteht darin, Rückbezüge zu ermöglichen, dh. die in ihnen eingeschlossenen Elemente zu einem späteren Zeitpunkt referenzieren zu können. Das klingt etwas abstrakt, das folgende Beispiel zeigt, worum es es geht.

In dem RA a[12]b taucht an 2. Stelle die Klasse [12] auf, d.h. hier kann eine 1 oder eine 2 stehen. Der o.a. RA bildet also zwei Zeichenketten ab: a1b und a2b. Angenommen, wir wollten nun ausdrücken, dass am Ende der Zeichenkette die gleiche Ziffer stehen soll, wie an der 2. Stelle (also a1b1 oder a2b2) – wie sollte das gehen? Wir wissen ja nicht, ob die 1 oder 2 an der 2. Stelle steht, und mit a[12]b[12] würde es deshalb nicht funktionieren, weil hiermit ja auch die Möglichkeiten a1b2 sowie a2b1 ausgedrückt sind.

Genau hier greift die Klammer, denn durch diese wird der geklammerte Ausdruck intern indiziert, was konkret bedeutet, dass ihm eine Zahl zugewisen wird, auf die man sich an anderer Stelle beziehen kann. Diese Indizierung beginnt mit '1', wenn weitere Klammern im RA auftreten, wird einfach hochgezählt. Durch die Klammerung a([12])b haben wir jetzt die Möglichkeit, das auszudrücken, was wir wollen, nämlich durch das Aufführen des Index des geklammerten Ausdruckes: a([12])b**\1**. Das bedeutet soviel wie 'welches Element auch immer des geklammerten Ausdrucks vertreten ist, das Element am Ende ist mit diesem identisch'.

Ein RA wie abc([1-3])def\1([4-6])ghi\2 könnte also als abc1def14ghi4, oder als abc2def24ghi4, oder als abc3def34ghi4 oder als abc1def15ghi5 usw. usf. realisiert sein. Auf jeden Fall aber sind das 4. und 8. sowie das 9. und 13. Element in der Zeichenkette jeweils identisch.

***Greedy* vs *Non-Greedy* RA**

Das mit 'gierig' zu übersetzende *greedy* bezieht sich im Rahmen der RA auf den Umstand, dass bei einem Mustervergleich RA—Zeichenkette, bei dem der RA nicht nur die Gesamtfolge, sondern auch Teilketten davon beschreibt, i.d.R. die größtmögliche bzw. längste der potentiellen Zeichenfolgenkandidaten mit dem RA in Übereinstimmung gebracht wird. Sehen Sie dazu das folgende Beispiel, in dem der Punkt '.' auftaucht, der in RA für "beliebiges Zeichen außer *neue Zeile*" steht.

Der RA <.*> findet sich *theoretisch* dreimal in der Zeichenkette <p>Hallo<\p>, nämlich in <p>, in <\p> und natürlich in der Gesamtzeichenkette. *Greedy* heißt nun, dass bei der konkreten Anwendung immer die längste dieser Möglichkeiten identifiziert wird, also im konkreten Fall die ganze Zeichenkette. Um bei Bedarf auszudrücken, dass z.B. nur das erste Spitzklammerpaar gemeint ist, müsste der Operator * als *non-greedy* ausgewiesen werden dadurch, dass ihm ein Fragezeichen nachgestellt wird: <.*?> würde jetzt, auf die o.a. Kette angewendet, nur das erste Spitzklammerpaar identifizieren.[5]

Wir wollen mit ein paar Beispielen schließen, die noch überschaubar und sprachtechnologisch etwas realistischer sind u.a. deshalb, weil sie im Rahmen von Taggern oder Chunk-Parsern eingesetzt werden können. Darin finden wir noch das Dollarzeichen, das eigentlich das Zeilenende markiert, im Rahmen der Tokenisierung allerdings das Tokenende.

RA im Rahmen von Regex-Taggern

<table>
<tr><td>([Tt]he|[Aa]n?)$</td><td>Identifiziert die Wörter The, the, An, an, A und a.</td></tr>
<tr><td>.*[aiu]ble$</td><td>Identifziert Wörter, die auf -able, -ible und -uble enden (häufig Adjektive).</td></tr>
<tr><td>.*(ness|hood)$</td><td>Identifiziert Wörter, die auf -ness oder -hood enden (typischerweise Nomina).</td></tr>
<tr><td>.*(ing|ed)$'</td><td>Identifiziert Wörter, die auf -ing oder -ed enden (häufig Verben)</td></tr>
</table>

RA im Rahmen eines Chunk-Parsers

<table>
<tr><td>(name/)|(det/((adv/)*(adj/))*noun/)</td><td>Identifiziert eine Tagkette vom Typ NP</td></tr>
<tr><td>verb/(np/|pp/)*</td><td>Identifziert eine Tagkette vom Typ VP</td></tr>
</table>

[5] Dieses ist nicht in allen Sprachen implementiert, in Python aber schon.