

# Segmentierung und Klassifizierung in der automatischen lexikalischen Analyse

Unter 'automatischer lexikalischer Analyse' ist ein Prozess zu verstehen, bei dem eine sprachliche Eingabekette durch entsprechende Programme zunächst in eine Reihe von Einheiten segmentiert wird und diese Einheiten anschließend mit Bezug auf bestimmte, für die Weiterverarbeitung relevante Information klassifiziert werden.

Das Modell in Abb.1 verortet exemplarisch die lexikalische Analyse im Rahmen eines Parsers, also eines Programmes zur automatischen Analyse von Sätzen:

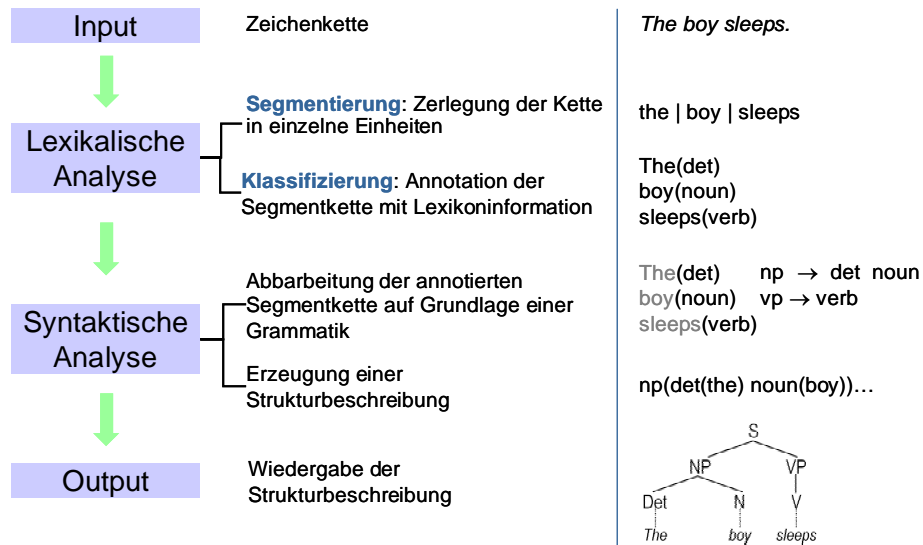


Abb.1: Verfahrensablauf eines Parsers (vereinfacht)

In der lexikalischen Analyse wird die Eingabekette 'The boy sleeps' zunächst in die Segmente 'the', 'boy' und 'sleeps' zerlegt; anschließend werden diese Segmente mit Bezug auf ihre jeweilige lexikalische Kategorie (Determinator, Nomen, Verb) klassifiziert. Output der lexikalischen Analyse wäre in diesem Fall eine Segmentkette wie in etwa 'the/DET', 'boy/N', 'sleeps/V'.

## Teil 1: Automatische Segmentierung

### Einleitung

Der Begriff 'Segmentierung' bezieht sich ganz allgemein auf die Unterteilung einer Einheit in eine Reihe von einzelnen Segmenten (lat. *segmentum* 'Abschnitt'). Bei sprachlichen Einheiten wie z.B. Laut- oder Wortketten ist die Segmentierung Grundvoraussetzung für jegliche Art der Weiterverarbeitung.

Bei der menschlichen Sprachverarbeitung läuft dieser Prozess i.d.R. unbewusst auf Grundlage der Kenntnis, die über die jeweilige Sprache vorliegt.

### Die Segmentierung gesprochener Sprache

Abb. 2 zeigt die Schallwellen, die bei einem akustischen Ereignis entstanden sind. Wie man sehen kann, liegt hier ein Lautkontinuum vor:



Abb. 2: Akustisches Kontinuum

Da es sich bei diesem akustischen Ereignis um gesprochene Sprache handelt, können wir es wie folgt phonetisch transkribieren:

halolibə:ʃtudi:vəndə

Abb. 3: Phonetische Transkription

Allerdings bedarf es der Kenntnis über die jeweilige Sprache, um das Kontinuum in Einheiten dieser Sprache zu zerlegen, wie z.B. die einzelnen Wörter, aus denen es sich konstituiert:

hallo—liebe—studierende

Abb. 4: Kette von Wörtern

Die in Abb. 3 durch Gedankenstriche markierten Grenzen zwischen den einzelnen Wörtern sind in der gesprochenen Form nicht ausmachbar.

Dieses Beispiel verweist auf das Hauptproblem maschineller Sprachverarbeitung, nämlich die Frage, wie das, was bei der menschlichen Sprachanalyse als mehr oder minder automatisches Resultat herauskommt, durch eine Maschine erzeugt werden kann, die ja – wenn überhaupt – höchstens in kleinen Teilen über das für die Analyse nötige Sprach- und Weltwissen verfügt.

Im Rahmen der Computerlinguistik und Sprachtechnologie lassen sich hier sehr grob drei Richtungen ausmachen:

- es werden – quasi ergebnisorientiert – Methoden eingesetzt, die zum 'korrekten' Resultat führen, aber ggf. nur wenig mit der menschlichen Sprachverarbeitung zu tun haben
- es wird versucht, das zu modellieren, was bei der menschlichen Sprachverarbeitung passiert.
- es wird eine Mischung aus beiden Richtungen verwendet

### Die Segmentierung geschriebener Sprache

Die beiden Kernfragen, die prinzipiell mit der automatischen Segmentierung sprachlicher Syntgmen verbunden sind, lauten:

- Welchem Zweck soll die Segmentierung dienen, und damit verbunden:
- In welche Einheiten sollen die Syntagmen zerlegt werden?

### Die Segmentierung von Wörtern

Mit Bezug auf die Kernfragen der Segmentierung können wir Wörter in unterschiedliche Segmente zerlegen, z.B. in die Buchstaben, aus denen sie sich zusammensetzen, in die (orthographischen) Silben, aus denen sie bestehen, oder in die Morpheme, die sie konstituieren. Ein Wort wie *international* würde mit Bezug auf diese Parameter wie folgt segmentiert:

1. i-n-t-e-r-n-a-t-i-o-n-a-l
2. in-ter-na-tio-nal
3. inter-nation-al

Diese drei Segmentierungen liegen auf unterschiedlichen Ebenen, nämlich Orthographie I: Grundeinheit einzelne Buchstaben, Orthographie II: Grundheit orthograpische Silben und Morphologie.

Die automatische Segmentierung von Wörtern ist für verschiedene Anwendungen relevant. So stehen z.B. in zahlreichen Textverarbeitungsprogrammen Module für die Silbentrennung zur Verfügung. Abb. 5 zeigt den Vorschlag für die Trennung des Wortes *Segmentierung* in MS Word:

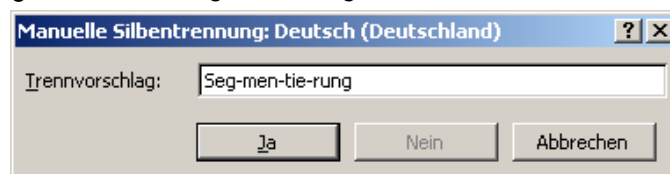


Abb. 5: Silbentrennung

Eine morphologische Segmentierung von Wörtern ist u.a. dann von Interesse, wenn es um die Erzeugung von Lexika oder Interlinearversionen fremdsprachiger Texte geht. Abb. 6 zeigt eine mit Shoebox erstellte Analyse eines Satzes. In der zweiten Zeile (Tag \m) ist zu sehen, wie die Wörter aus der ersten Zeile (Tag \t) automatisch auf der Basis einer entsprechenden Morphemdatenbank zerlegt werden:

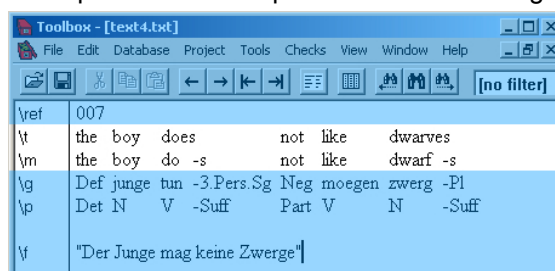


Abb. 6: Morphologische Analyse mit Shoebox

### Automatische Silbentrennung

Die automatische Silbentrennung ist nicht trivial. Dieses liegt zum einen daran, dass der Input, der segmentiert werden soll, häufig nicht in der Form vorliegt, auf die sich die Einheit 'Silbe' bezieht. Silben können beschrieben werden als...

...a unit of pronunciation that consists of a **central peak of sonority** (usually a vowel or a diphthong, abbreviated as **V**) which may be surrounded by one or more consonants (abbreviated to **C**) that **cluster** around it. The kernel or centre of the syllable, ie. V, is the so-called **nucleus**, those sounds preceding it are called **onset** and those that follow form the **coda** (S. Hackmack: *Introducing Linguistics: A Basic Course*. 2005)

Silbe	Muster	Onset	Nukleus	Koda
<i>in</i>	VC		ɪ	n
<i>die</i>	CV	d	i:	
<i>Tipp</i>	CVC	t	ɪ	p
<i>Trip</i>	CCVC	tʁ	ɪ	p
<i>Strom</i>	CCCVC	ʃtʁ	o:	m
<i>Trotz</i>	CCVCC	tʁ	ɔ	ts

Abb. 7: Analyse von Silben

Ein bei der menschlichen Silbentrennung ganz automatisches Verfahren verläuft so, dass das zu segmentierende Wort quasi im Kopf 'vorgelesen' und auf die dabei entstehende Lautkette dann das internalisierte Konzept 'Silbe' angewendet wird.

Bei der automatischen Silbentrennung in einem Textverarbeitungsprogramm liegt das zu analysierende Wort allerdings nicht in akustischer Form vor, sondern als Kette von Buchstaben, dh. es müssen andere Strategien angewendet werden, um korrekt zu segmentieren. Bei diesem Prozess kommt hinzu, dass einzelsprachenspezifische Regeln zur Silbentrennung zu berücksichtigen sind, die eine gegebene Buchstabenkette in manchen Fällen eben nicht in Gänze in Sprachsilben zerlegen. Solche Regeln sind z.T. in Eselbrücken wie der folgenden formuliert: "Trenne niemals s und t, denn das tut den beiden weh".<sup>1</sup>

Der folgende Ablauf zeigt zwei mögliche Verfahrensschritte eines Silbentrennprogrammes. Input ist jeweils WORT, welches ggf. in mehreren Schritten durchlaufen wird. Im ersten Verfahren geht es um die Abtrennung von Wortteilen am Wortanfang; im zweiten Verfahren um Abtrennungen am Wortende. Wie diese Verfahren genau implementiert werden, hängt von der jeweiligen Programmiersprache ab, in der die Textverarbeitungsprogramme geschrieben sind.

**Prozess: Iterative Abtrennung der Präfixe**

Restriktion: Längstes Präfix zuerst

Voraussetzung: 1. WORT liegt als Buchstabenkette vor  
 2. Gespeicherte Liste von Präfixen, wie z.B. [ab-, auf-, de-, des-, ein-, einzel-, ge-, in-, inter-, un-, unter-, vor ]

Erläuterung: Bei dieser Anweisung besteht ein Bezug zur Morphologie (s.u.), der – wie weiter oben gesehen – bei der Silbentrennung nicht durchgängig vorhanden ist. Um diese Anwendung durchführen zu können, muss das Programm auf die Liste von Präfixen zurückgreifen können. WORT kann dann mit Bezug auf diese Liste analysiert werden, beispielsweise in Form eines Mustervergleiches (informell: steht das Präfix xyz am Wortanfang? Wenn ja, schneide es ab). Dabei würden – der Restriktion entsprechend – immer zunächst die längsten Präfixe verarbeitet. 'Iterativ' (lat. *iterare*: "wiederholen") bedeutet, dass dieser Prozess auf sein Ergebnis erneut angewendet wird.

Beispiele: WORT: international  
 Präfixliste: in- inter-  
 Restriktion: Längstes Präfix zuerst  
 Ergebnis: inter-national  
 WORT: desinformation  
 Präfixliste: de- des-  
 Restriktion: Längstes Präfix zuerst  
 Ergebnis<sub>a</sub>: des-information  
 WORT: information  
 Präfixliste: in-  
 Ergebnis<sub>b</sub>: in-formation  
 Ergebnis<sub>a+b</sub>: des-in-formation

**Prozess: Abtrennung der Suffixe**

Restriktion: Wenn WORT die Endung -ung aufweist, trenne vor dem Konsonanten, der unmittelbar vor der Endung steht.

Voraussetzung: 1. WORT liegt als Buchstabenkette vor  
 2. WORT ist mit Bezug auf konsonantische Buchstaben (k) und vokalische Buchstaben (v) analysiert (kv-WORT)  
 3. Gespeicherte Liste von Suffixen, wie z.B. [-ant, -heit, -ion, -keit, -ung]

Erläuterung: Bei dieser Anweisung besteht prinzipiell ein Bezug zur Morphologie (s.u.), der allerdings

<sup>1</sup> Dieses ist eine Regel aus der Zeit vor der Rechtschreibreform. Dieser Text - wie auch alle anderen Texte von mir - ist nicht dem Diktat der neuen Rechtschreibung unterworfen, sondern pickt sich aus den jeweiligen Versionen das heraus, was mir relativ sinnvoll erscheint.

für bestimmte Suffixe aufgehoben wird. Um diese Anwendung durchführen zu können, muss das Programm auf die Liste von Suffixen zurückgreifen können. WORT würde dann – analog zur Abtrennung der Präfixe – in einem ersten Durchgang durchlaufen und mit Bezug auf die Suffixliste analysiert. Wenn dabei die Endung -ung ermittelt ist, würde die Trennung entsprechend der o.a. Restriktion vorgenommen, was bei wie einem Wort wie *Bildung* dazu führt, dass es nicht in die Morpheme Bild- und -ung, sondern in die Silben Bil- und -dung segmentiert wird.

Beispiele:

WORT:	schönheit
Suffixliste:	-heit
Ergebnis:	schön-heit
WORT:	handlung
kvWORT:	KVKKKVKK
Suffixliste:	-ung
Restriktion:	Trenne vor Konsonanten, der vor -ung steht
kvWORT:	KVKK-KVKK
Ergebnis:	hand-lung

Es gibt natürlich eine Vielzahl weiterer Verfahren, z.B. solchen, die der o.a. Eselsbrücke "trenne niemals s und t" Genüge tun (und dazu dann natürlich entsprechende Ausnahmen, vgl. Mei-ster vs \*Hau-stür, Haus-tür), für die Illustration der automatischen Silbentrennung aber brauchen diese nicht alle einzeln erläutert zu werden.

### Automatische Morphemanalyse

Bei der automatischen Morphemanalyse geht es darum, dass ein eingegebenes Wort mit Bezug auf die Morpheme, die es enthält, segmentiert wird.<sup>2</sup>

Der Begriff 'Morphem' hat in der Sprachwissenschaft eine lange Tradition; sein häufiger Gebrauch darf dabei nicht darüber hinwegtäuschen, dass er alles andere als einheitlich definiert ist und dass es zahlreiche theoretische Ansätze gibt, die dieses Konstrukt aus guten Gründen als nicht geeignet für die Beschreibung tatsächlicher morphologischer Prozesse erachten. Diese Frage wird hier nicht vertieft, d.h. wir arbeiten mit einer traditionellen Definition von Morphem, nach der ein Morphem eine Klasse äquivalenter Morphe ist, die es in verschiedenen Umgebungen repräsentieren. Ein Morph kann als ein rekurrentes Minimalzeichen beschrieben werden.<sup>3</sup>

Eine automatische Morphemanalyse setzt voraus, dass dem Programm eine wie auch immer geartete Kenntnis der in der Sprache vorhandenen morphologischen Regeln zur Verfügung steht.

Manche Programme, wie z.B. das oben bereits erwähnte *Shoebox*, enthalten in ihrer Wissensbasis Morphemdatenbanken bzw. -lexika, die dann als Grundlage beim Parsen bzw. bei der Analyse eines Wortes eingesetzt werden. Vereinfacht kann das Prozedere wie folgt dargestellt werden:

<b>Wissenbasis:</b>	
Freie Morpheme:	dog, girl, walk, the, develop, fair
Suffixe:	-s, -ed, -ment, -al
Affixe:	in-, dis-, un-
<b>Eingabewörter:</b>	dogs, unfair, development
<b>Ergebnis:</b>	dog-s, un-fair, develop-ment-al

Andere Programme versuchen, bei der morphologischen Analyse bestimmte Algorithmen anzuwenden, die die Eingabekette z.B. auf der Basis (a) ihrer orthographischen Form (bzw. deren Repräsentation als Konsonant-Vokal-Ketten) und (b) bestimmter für die Sprache geltenden Segmentierungsregeln zu analysieren.

Das klingt ein wenig schwammig, das folgende Beispiel soll dazu dienen, diese Aussagen zu konkretisieren. Dazu betrachten wir den sog. Porter-Stemmer-Algorithmus etwas näher.

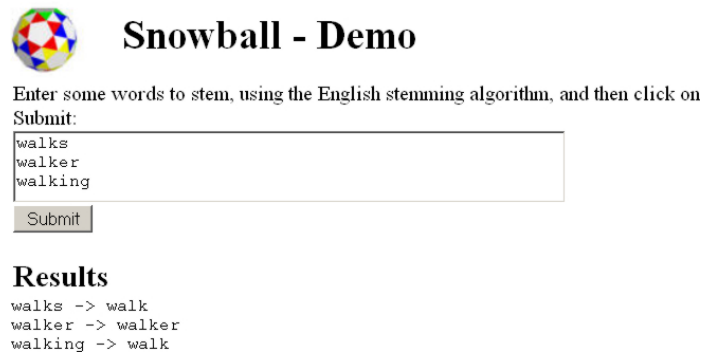
Vorwarnend sei an dieser Stelle erwähnt, dass die bei der Beschreibung des Programmes verwendete Terminologie z.T. erheblich von der Begrifflichkeit abweicht, die im Rahmen der linguistischen Phonologie und Morphologie eingesetzt wird. Auf dieses Problem wird an den relevanten Stellen aber hingewiesen.

<sup>2</sup> Ein umständlichere, aber präzisere Ausdrucksweise wäre: mit Bezug auf die Morpheme, die die Allomorphe, aus denen sich das Wort konstituiert, jeweils repräsentieren.

<sup>3</sup> Für eine Anfängereinführung in die Morphologie und die entsprechende Terminologie lesen Sie bitte den Text 'The complex linguistic sign I', den ich Ihnen entweder mitbringe oder über meine Webpage verlinke

## Die Grundzüge des Porter-Stemmer-Algorithmus

Ganz allgemein gesprochen ist ein Stemmer ein Programm, das versucht, bei einem Wort Affixe 'abzuschneiden'. Nämlich den Ausdruck 'stem' wörtlich, wäre damit ein Programm gemeint, das ausschließlich Flexionsaffixe abschneidet und den Stamm zurückgibt, wie z.B. in Schönheiten → Schönheit. Wenn wir die Wörter *walks*, *walker* und *walking* dem Porter-Stemmer-Algorithmus unterziehen, erhalten wir entsprechend folgendes Resultat:



**Snowball - Demo**

Enter some words to stem, using the English stemming algorithm, and then click on Submit:

walks  
walker  
walking

Submit

**Results**

walks -> walk  
walker -> walker  
walking -> walk

**Abb. 6: Porter-Stemmer**

Allerdings ist bei vielen Stemmern, so auch dem Porter-Stemmer, zu beobachten, dass keine klare Trennung zwischen Basis, Stamm und Wurzel vorgenommen wird. Martin Porter, der Entwickler des gleichnamigen Stemmers, gibt dazu Folgendes zu bedenken:

One can make some distinction between *root* and *stem*. [...]. But here we will think of the stem as the residue of the stemming process, and the root as the inner word from which the stemmed word derives, so we think of root to some extent in an etymological way. It must be admitted that when you start thinking hard about these concepts *root*, *stem*, *suffix*, *prefix* ... they turn out to be very difficult indeed to define. Nor do definitions, even if we arrive at them, help us much. After all, suffix stripping is a practical aid in IR [Information Retrieval, S.H.], not an exercise in linguistics or etymology. (Martin Porter: *Snowball: A language for stemming algorithms*. <http://snowball.tartarus.org/texts/introduction.html>, Accessed: 08.08.08)

Am letzten Satz dieses Zitats werden die weiter oben diskutierten Kernfragen der Segmentierung wieder sehr sichtbar und Porter gibt mit Bezug auf seinen Stemming-Algorithmus auch eine deutliche Antwort.

Zweck der Segmentierung im Porter Stemmer ist die Weiterverarbeitung im Rahmen des *Information Retrieval*, dh. es kommt nicht so sehr auf eine linguistisch informierte morphologische Analyse an, sondern darauf, Bezüge zwischen einzelnen Wörtern zu erstellen derart, dass z.B. eine semantische Textindizierung ermöglicht wird. So erzeugt das Entfernen der Suffixe in den Wörtern *beauties*, *beautiful* und *beautifully* mit dem Porter-Stemmer-Algorithmus jeweils dasselbe Ergebnis, nämlich 'beauti'.

An diesem Beispiel ist erkennbar, dass nicht nur Flexions-, sondern auch bestimmte Derivationsuffixe abgeschnitten werden. Man sieht, dass die Begriffe wie 'Suffix' und 'Stamm' im Porter-Stemmer anders und weniger präzise gebraucht werden als in der linguistisch orientierten Morphologie.

Wie aber funktioniert dieser Algorithmus? Eine detaillierte Beschreibung kann an dieser Stelle nicht geliefert werden, dh. es werden nur die elementaren Grundzüge des Stemmers für das Englische besprochen. Es gibt hier auf jeden Fall deutliche Analogien zu den weiter oben beschriebenen Silbentrennprogrammen.

### Die Grundeinheiten des Porter-Stemmers

Grundeinheit des Porter-Stemmer-Algorithmus sind Ketten bestehend aus Elementen der Mengen *v* (Vokal) und *c* (Konsonant).

Diese Mengen können wie folgt definiert werden:  $v = [A, E, I, O, U]$ .  $k =$  alle anderen.<sup>4</sup> Besonderheit: der Buchstabe 'y', welcher eigentlich ein Vokal ist, aber zu den Konsonanten gerechnet wird, wenn er nicht selber auf einen Konsonanten folgt. Nach diesem Schema sind die folgenden Abbildungen möglich:

roll: *cvcc*  
strings: *cccvc*  
under: *vccvc*  
fail: *cvvc*  
try: *cvv*  
boy: *cvc*

An diesen Ketten wird sehr deutlich, dass es sich bei den Konzepten *c* und *k* nicht – wie weiter oben bei der Beschreibung von Silben – um Klassen von Lauten handelt, sondern um Klassen von Buchstaben.

In vielen Wörtern finden wir Aneinanderreihungen von *c* bzw. *v*. Bei dem Wort *strings* zum Beispiel haben wir es mit drei aufeinanderfolgenden *c*, einem *v* und zwei aufeinanderfolgenden *c* zu tun. Derartige nicht-leere

<sup>4</sup> Die Eingabeketten werden offensichtlich normalisiert, dh. in Großbuchstaben umgewandelt.

Listen von *c* bzw. *v* werden durch die jeweiligen Großbuchstaben C und V repräsentiert, d.h. sowohl [ccc] und [cc] entsprechen C und das Wort *strings* wird wie folgt notiert: CVC.

Man sieht, dass C und V im Porter Stemmer etwas gänzlich anderes sind als in der Phonologie. Die Porter-Stemmer Analyse und die phonologische Analyse von *string* vergleichen sich wie folgt:

Porter-Stemmer-Grundeinheit: Buchstabenkette	Phonologische Grundeinheit: Lautkette
s-t-r-i-n-g-s → [ccc][v][cc] → CVC	stɪŋs → CCCVCC

**Abb. 7: Grundheiten im Porter Stemmer und in der Phonologie**

Im Porter-Stemmer wird mithin jedes Wort bzw. jeder Wortteil über eine Verkettung von C und V repräsentiert, wobei immer ein C auf ein V folgen muss bzw. umgekehrt – eine Kette wie \*CCVVCV ist danach ausgeschlossen.

Für Anfang und Ende dieser Ketten sind nur die folgenden 4 Kombinationen denkbar:

1. C – VCVC... – C z.B. but (CVC), falling (CVCVC) computer (CVCVCVC)
2. C – VCVC... – V z.B. try (CV), faulty (CVCV), mistletoe (CVCVCV)
3. V – CVCV... – C z.B. eat (VC), impress (VCVC), uncertain (VCVCVC)
4. V – CVCV... – V z.B. intro (VCV), unhappy (VCVCV), uncertainty (VCVCVCV)

Zwischen den jeweiligen Anfangs- und Endelementen können dabei VC- bzw. CV Aneinanderreihung beliebiger Länge auftreten. Diese Ergebnisse können wie folgt in einer Formel zusammengefasst werden:

$$[C] (VC)^m [V]$$

**Abb. 8: Porter-Stemmer Formel**

In dieser Formel stehen die eckigen Klammern für Fakultativität. Das hochgestellte 'm' gibt die Anzahl der Wiederholungen der in der Klammer aufgeführten VC wieder. Der Wert von m ist  $\geq 0$ , wobei 0 'kein VC' bedeutet, 1 'ein VC' usw. Um diese Formel zu illustrieren, werden wir die Wörter *but*, *patient*, *computer*, *into*, *improbable*, *mistletoe*, *uncertain* und *uncertainties* mit ihr beschreiben.

1. but: CVC =  $C(VC)^1$
2. patient: CVCVC =  $C(VC)^2$
3. computer: CVCVCVC =  $C(VC)^3$
4. into: VCV =  $(VC)^1V$
5. improbable: VCVCVCV =  $(VC)^3V$
6. mistletoe: CVCVCV =  $(CV)^2$
7. uncertain: VCVCVC =  $(VC)^3$
8. uncertainties: VCVCVCVC =  $(VC)^4$

**Die Regeln des Porter-Stemmers**

Die für das Abschneiden der Suffixe formulierten Regeln sind alle nach dem folgenden Prinzip aufgebaut:

$$(Bedingung) S_1 \rightarrow S_2$$

**Abb. 9: Grundform einer Porter-Stemmer Regel**

Diese Regel ist wie folgt zu lesen: Wenn ein Wort im Suffix  $S_1$  endet und die in der Regel formulierte Bedingung erfüllt, dann ersetze in diesem Wort  $S_1$  durch  $S_2$ .

Eine Konkretisierung dieser Regel könnte z.B. wie folgt lauten:

$$(m>0) EED \rightarrow EE$$

Die Substitution besagt, dass das Suffix -eed durch das Suffix -ee ersetzt wird, die Bedingung drückt aus, dass diese Regel nur auf Stämme anwendbar ist, deren Wert für m größer als 0 ist. Für die Wörter *agreed* und *feed* würde entsprechend folgendes Ergebnis erzielt:

agreed → agree (agreed =  $(VC)^1$ , die Bedingung ist erfüllt)

feed → feed (feed =  $(VC)^0V$ , die Bedingung ist nicht erfüllt)

Natürlich kann ein Suffix auch durch  $\emptyset$  substituiert werden, z.B. in der folgenden Regel, die bei bestimmten Stämmen das -er abschneidet:

$$(m>1) ER \rightarrow$$

Diese Regel wäre z.B. auf das Wort *airliner* anwendbar: als  $(VC)^2$  erfüllt *airliner* die Bedingung. Die Wörter *stalker* oder *walker* enden auch auf -er, aber sie erfüllen die Bedingung nicht, da sie jeweils die Form  $C(VC)^1$  aufweisen:

airliner → airlin

stalker → stalker

Die Bedingungen können auch komplexer sein als in unserem Beispiel, wobei die einzelnen Teilbedingungen durch die booleschen Operatoren 'und', 'nicht' und 'oder' verknüpft werden:



Bedingung	Erläuterung
(m>1 and (*S or *T))	Stamm weist m>1 auf und endet auf entweder S oder T
(m< 1 and not (*L or *R))	Stamm weist m>1 auf und endet nicht auf L oder R
(*d and not (*L or *S or *Z))	Stamm endet auf einen Doppelkonsonanten (*d), aber nicht auf L, S oder Z
(m=0 and *v*)	Stamm weist m=0 auf und enthält einen Vokal

Abb. 10: Bedingungen in einer Porter-Stemmer Regel

Die nachstehende Tabelle zeigt exemplarisch, wie der Porter Stemmer auf Grundlage einer mit der Formel [C] (VC)<sup>m</sup> [V] analysierten Buchstabenkette und den Regeln zu seinen Substitutionsresultaten kommt:

Wort	[C] (VC) <sup>m</sup> [V]	Regel	Substitution
goodness	C(VC) <sup>2</sup>	(m>0) NESS →	good
replacement	C(VC) <sup>4</sup>	(m>1) EMENT→	replac
adoption	(VC) <sup>3</sup>	(m>1 and *S or *T) ION →	adopt
formalize	C(VC) <sup>3</sup> V	(m>0) ALIZE → AL	formal
operator	(VC) <sup>4</sup>	(m>0) ATOR → ATE	operate

Abb. 11: Anwendung der Porter-Stemmer-Regeln

Im Porter Stemmer nun sind sogenannte Regel-Sets implementiert, die jeweils Mengen von Regeln enthalten. Für ein bestimmtes Input-Wort sind nicht alle Regeln relevant, doch werden bei der konkreten Verarbeitung sukzessive alle Sets durchlaufen und es wird geprüft, ob eine Regel anwendbar ist, oder nicht. Dieses sehen wir am folgenden Beispiel:

Wort	[C] (VC) <sup>m</sup> [V]	Regel	Substitution
digitization	C(VC) <sup>5</sup>	(m>0) IZATION → IZE	digitize
digitize	C(VC) <sup>3</sup> V	(m>1) IZE →	digit

Abb. 12: Sukzessive Anwendung der Porter-Stemmer-Regeln

### Die Segmentierung von Sätzen

Auch bei dieser Aufgabe stellen sich wieder die Kernfragen der Segmentierung, die hier zur Erinnerung noch einmal aufgeführt sind:

- Welchem Zweck soll die Segmentierung dienen und – damit verbunden -
- In welche Einheiten sollen die Syntagmen zerlegt werden?

In **Abb. 1** haben wir zwei mögliche Antworten auf diese Frage gesehen: die Eingabekette, der Satz *The boy sleeps*, wird in Schritt 1 der lexikalischen Analyse in eine Reihe von Wörtern zerlegt, die dann, bei der Klassifizierung, ihrer jeweiligen Wortart zugewiesen werden. Auf diese Weise wird die Eingabekette für einen bestimmten Zweck, nämlich die Weiterverarbeitung in der syntaktischen Komponente, aufbereitet. In der Tat ist diese Art der Segmentierung, also die Zerlegung eines Textes in seine Wörter, häufig Grundvoraussetzung für jegliche Art der Weiterverarbeitung.

Auf unser Beispiel bezogen scheinen wir es hier mit einer trivialen Aufgabe zu tun zu haben: die einzelnen Segmente von *The boy sleeps* sind ja klar auszumachen, da die Grenzmarkierungen zwischen ihnen offenkundig sind: die Wörter sind jeweils durch Leerzeichen voneinander getrennt.

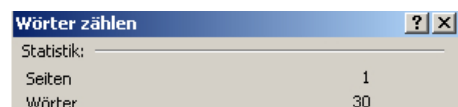
Leider aber ist die Geschichte alles andere als trivial und tangiert ganz unmittelbar eines der Kernprobleme der Sprachwissenschaft, nämlich die Frage: was zählt in einem Text überhaupt als Wort?

Nehmen wir zur Illustration des Problem es den folgenden Text:

Die Eislawine riss Haken und Seile an der schwierigsten Passage mit sich und ließ die Bergsteiger, die den Aufstieg am K2 bei kommerziellen Anbietern gebucht hatten, verwirrt und hilflos zurück.  
 (<http://www.spiegel.de/reise/aktuell/0,1518,570132,00.html>, Accessed 06. 08. 08)

Wenn verschiedene Menschen aufgefordert werden, die Wörter in diesem Text zu zählen, werden interessanterweise unterschiedliche Ergebnisse erzielt. Als mögliche Antworten werden genannt: 26, 29, 30, 31 und 32. Diese Antworten hängen ab davon, was jeweils als Wort gilt – und was nicht.

- Zählen wir alle Elemente als Wort, die durch Leerzeichen begrenzt sind, erhalten wir 30. So verfährt z.B. MS Word:
- Lösen wir dagegen alle Komposita im Text auf (*Eislawine* = *Eis* und *Lawine*), erhielten wir 32 Wörter
- Tun wir das nicht, und zählen wir auch die Elemente *zurück* und *ließ* als ein Wort (als Verb mit abtrennbarem Präfix *zurücklassen*), so landen wir bei 29
- Den Wert 26 schließlich erzielen wir, wenn wir alles als Wort ansehen, was durch Leerzeichen begrenzt ist, aber das dreimalige Vorkommen von jeweils *und* und *die* nur einmal zählen.



Was dieses Beispiel zeigt, ist die konzeptuelle Unsicherheit, die mit dem Begriff 'Wort' verbunden ist. Für den letztgenannten Fall, also die Frage, ob die Elemente *und* und *die* im Beispieltext jeweils mit Bezug auf ihre Vorkommensfrequenz gezählt werden, oder nicht, bietet sich die in der Linguistik auf allen Ebenen der Beschreibung verbreitete Type-Token-Distinktion zur Klärung.

**Types und Tokens**

The distinction between a type and its tokens is an ontological one between a general sort of thing and its particular concrete instances (to put it in an intuitive and preliminary way). So for example consider the number of words in the Gertrude Stein line from her poem Sacred Emily on the page in front of the reader's eyes: Rose is a rose is a rose is a rose.

In one sense of 'word' we may count three different words; in another sense we may count ten different words. C. S. Peirce called words in the first sense "types" and words in the second sense "tokens". Types are generally said to be abstract and unique; tokens are concrete particulars, composed of ink, pixels of light (or the suitably circumscribed lack thereof) on a computer screen, electronic strings of dots and dashes, smoke signals, hand signals, sound waves, etc.

(<http://plato.stanford.edu/entries/types-tokens/>, Accessed 06.08.08)

Wie schon gesagt findet sich die Unterscheidung zwischen (a) einer abstrakten Klasse und (b) den konkreten Instanzen bzw. Vorkommen, die diese Klasse in verschiedenen Umgebungen repräsentieren, auf allen Ebenen und diversen Abstraktionsstufen der linguistischen Beschreibung. Beispiele hierzu sind:

Ebene	Abstrakte Klasse	Instanzen / Vorkommen
Orthographie	'und'	und und und <b>und</b> und...
Phonologie	/ʌ/	[ʌ] [tʰ]
Morphologie	{Plural}	[s] [z] [ɪz]
Lexikologie	WALK	walk, walks, walked, walking
Syntax	NP	the boy, John, a girl with kaleidoscope eyes...

**Abb. 13: Type-Token Distinktion auf verschiedenen Abstraktionsebenen der Linguistik**

Nach dieser Erläuterung wird klar, dass wir es mit Bezug auf die Elemente *und* und *die* im Beispieltext auf der Vorseite mit zwei Types zu tun haben, die jeweils durch drei Tokens realisiert sind.

Programme, die im für einen gegebenen Text eine der automatische lexikalischen Analyse vollziehen, werden, Tokenizer (dt: Tokenisierer) genannt.

**Tokenizer**

Ein Tokenizer ist ein ein Programm, welches auf der Basis bestimmter Vorgaben einen Input in einzelne Tokens zerlegt. Für uns interessant sind nur diejenigen Tokenizer, die als Input einen Text bekommen und als Output diesen Text in einzelne Tokens zerlegen

Was jeder Tokenizer braucht, sind Verfahren, um die Grenzen zwischen einzelnen Tokens erkennen zu können. Um das zu erzielen, muss wiederum klar sein, was mit 'Token' jeweils gemeint ist, und dieses hängt natürlich ab davon, für welchen Zweck ein Text tokenisiert wird.

Im folgenden sind informell zwei unterschiedliche Arten von Weiterverarbeitung beschrieben, die jeweils andere Sorten von Element als Token benötigen. Wir gehen von folgenden Text aus:

*Der Junge zerschlägt den Krug und eine Vase*

1. Weiterverarbeitung in einem Parser  
Hier müssten alle Wortformen, die den Text konstituieren, tokenisiert werden:  
Der-Junge-zerschlägt-den-Krug-und-eine-Vase
2. Semantische Textindizierung im Rahmen des *Information-Retrieval*  
Hier müssten nur Inhaltswörter tokenisiert werden, die sogenannten *Stoppwörter* (definite und indefinite Artikel, Konjunktionen, bestimmte Präpositionen etc.) können ignoriert werden:  
Junge-zerschlägt-Krug-Vase

Während in also im ersten Fall die Grenzmarkierungen zwischen den Tokens tatsächlich den Leerzeichen zwischen den Wortformen entsprechen, wären es im zweiten Fall sowohl Leerzeichen als auch Elemente aus einer vordefinierten Liste von Stoppwörtern.

Im Rahmen von Python steht uns mit der `.split`-Methode bereits ein rudimentärer Tokenizer zur Verfügung. Diese Methode wird auf Zeichenketten angewendet und liefert ohne weitere Parameter eine Liste von Tokens, die auf Grundlage der sog. *Whitespaces* erzeugt wurde. Unter 'Whitespace' fallen bei Python Leerzeichen, Absatzmarken und Tabulatoren:

```
>>> text = "der junge zerschlaegt den krug"
>>> text.split()
['der', 'junge', 'zerschlaegt', 'den', 'krug']
```

**Abb. 14: Whitespace-Tokenizer mit der `.split()` Methode in Python**



In die Argumentklammer hinter dem Funktionsnamen können allerdings Parameter angegeben werden, die eine andere Grenzziehung zur Folge haben. Das nächste Beispiel zeigt die Zeichenkette 'ab.cde.fgh.ijk', die im ersten Fall über Whitespaces, im zweiten Fall über das Interpunktionszeichen '.' tokenisiert wurde:<sup>5</sup>

```
>>> text = 'ab.cde.fgh.ijk'
>>> text.split()
['ab.cde.fgh.ijk']
>>> text.split('.')
['ab', 'cde', 'fgh', 'ijk']
```

Abb. 15: `.split()` Methode in Python ohne und mit Parameter

### Probleme der Tokenisierung

In den Beispielketten, die wir bis jetzt im Rahmen der Tokenisierung behandelt haben, traten keine größeren Probleme auf, d.h. die Tokens konnten relativ eindeutig identifiziert werden. Das ist allerdings häufig nicht der Fall, wie die folgende (unvollständige) Liste von Problemen zeigt, mit dem der erste Teil dieses Texts schließt:

**Problem 1** Was passiert in Fällen, in denen zwei distinkte Elemente auf ein einziges Token abgebildet werden sollen, wie z.B. bei bestimmten Ortsnamen oder englischen Komposita?

Beispiele: *New York, Brandenburger Tor, short story* (Kurzgeschichte), *sports utility vehicle* usw.

**Problem 2** Wie ist – ganz allgemein – mit Fließkommazahlen, Abkürzungen, zusammengesetzten Wörtern oder festformatierten Textteilen wie Mailadressen, Autokennzeichen oder ähnlichem zu verfahren?

Beispiele: *1.523,87 / FrI. Meier, Dr. Krause / hackmack@uni-bremen.de / HB-GH 236 / L'Orchidée*

**Problem 3** Mit Bezug auf Problem 2: wie wird der Umstand behandelt, dass bei Interpunktionszeichen quasi eine Informationsüberfrachtung zu beobachten ist; ein- und dasselbe Zeichen also in verschiedenen Kontexten unterschiedliche Information kodiert?

Beispiele: *FrI., z.B., etc., vgl., u.a.* Punkt als Abschlusszeichen einer Abkürzung<sup>6</sup>  
*Er kam. Er sah. Er siegte* Punkt als Satzgrenzenmarkierung  
*1.500,00 2.456.787,00* Punkt als Gruppierungszeichen für 3 Ziffern  
*hackmack@uni-bremen.de* Punkt als Trennzeichen in einer Pfadangabe

**Problem 4** Welche einzelsprachlichen Regeln und Konventionen bezüglich der Orthographie bzw. allgemein des Aufbaus des Schriftsystems müssen bei der Tokenisierung berücksichtigt werden?

Beispiele: Im Deutschen werden, wie in Problem 3 gesehen, Dezimalzahlen mithilfe von Punkten und Kommata strukturiert:

'Eintausendfünfhundert' : 1.500,00

Im Englischen wäre die Anordnung von Punkt und Komma genau andersherum: 1,500.00

Das Deutsche hat eine alphabetisch-orientierte Orthographie, d.h. die Grundeinheiten, auf die sich die Schriftzeichen beziehen, entsprechen in etwa den Phonemen und also der Ausdruckseite sprachlicher Zeichen. Andere Sprachen, wie z.B. das Chinesische, haben eine logographisch-orientierte Orthographie, d.h. die Grundeinheiten, auf die sich die Schriftzeichen beziehen, entsprechen in etwa semantischen Basiskonzepten und also der Inhaltsseite sprachlicher Zeichen

**Problem 5** Ist Groß- bzw. Kleinschreibung relevant oder nicht?

Beispiele: Im deutschen *Man sagt, dass Fliegen schnell fliegen* wird die kategoriale Ambiguität von 'fliegen' durch die Groß- bzw. Kleinschreibung aufgehoben: Alle Substantive werden groß geschrieben, ergo ist 'Fliegen' ein Substantiv und 'fliegen' ein Verb.

Im Englischen *Time flies like an arrow and fruit flies like a banana* (courtesy of Groucho Marx) ist die Differenzierung von 'flies' als Verb oder Substantiv nicht über die Groß- bzw. Kleinschreibung zu ermöglichen

<sup>5</sup> Für genauere Information über Tokenisierung in Python siehe den Text 'Tokenisierung'

<sup>6</sup> Die Auflösung dieser Abkürzungen ist auch nicht gerade trivial: in 'vgl.' hätten wir ein Token 'vgl' (welches dann, bei entsprechendem Wörterbuch, auf 'vergleiche' abgebildet werden könnte). In 'u.a.' müssten wir zwei Tokens ermitteln: 'unter' und 'anderem'.

## Teil 2: Automatische Klassifizierung

### Einleitung

Ganz allgemein gesprochen spielen Klassen in der Sprachwissenschaft eine zentrale Rolle. Wir verwenden Klassen auf jeder Ebene der sprachlichen Beschreibung: Klassen von Lauten (z.B. Bilabial oder Plosiv), Klassen von Morphen (z.B. Pluralsuffix oder Diminutivaffix), Klassen von Wörtern (z.B. Nomen oder Adjektiv), Klassen von Konstituenten (z.B. NP oder VP), Klassen von Konzepten (z.B. Bewegungsverb oder Lokalpräposition) usw.

Der Begriff 'Klasse', der synonym mit dem Begriff 'Kategorie' verwendet wird, steht dabei für eine Menge von Objekten oder Elementen (konkret oder abstrakt), die aufgrund bestimmter gemeinsamer Merkmale zusammengefasst werden können.

Klassifizierung bedeutet danach die Zuweisung eines Objektes zu seiner jeweiligen Klasse.

### Die automatische Klassifizierung von Wörtern zu Wortarten: POS-Tagger

#### Tag / Tagger / Tagging

Ganz allgemein gesprochen ist ein Tag so etwas wie ein Etikett oder Anhängsel, mit dem ein bestimmtes Objekt annotiert wird. Was dieses Objekt ist, und womit es annotiert wird, ist zunächst einmal völlig offen. In dem folgenden, frei erfundenen Beispiel wurden die Eigennamen mit Bezug auf die Frage, ob sie auf ein weibliches oder ein männliches Wesen verweisen, handgetaggt:

Karla<sub>FRAU</sub> und Klaus<sub>MANN</sub> können Lisa<sub>FRAU</sub> und Lothar<sub>MANN</sub> nicht leiden und fahren deshalb zu Beate<sub>FRAU</sub>.

Im nachstehenden Screenshot kann man sehen, wie Tags bei einem HTML-Dokument eingesetzt werden, um bestimmte Textauszeichnungen zu erzeugen:

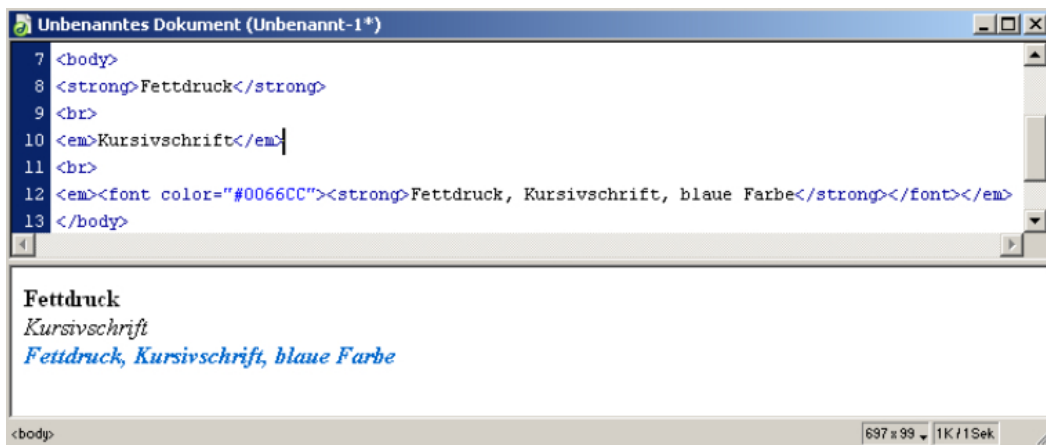


Abb. 16: HTML-Tags

Im oberen Teil des Screenshots sieht man den Quellcode. Darin sind die auszuzeichnenden Textstellen durch die Tags `<strong></strong>` (für Fettdruck), `<em></em>` (für 'emphasis', was auf Kursivschrift abgebildet wird) und `<font color=""></font>` für die Schriftfarbe umschlossen. Mit diesen Tags sind bestimmte Prozeduren verbunden, die das Ausgabeformat beeinflussen. Dieses ist auf dem unteren Teil des Fensters, der die Monitor- bzw. Druckansicht wiedergibt, zu sehen.

Ein Tagger nun ist ein Programm, welches jedem Element einer tokenisierten Eingabekette auf Grundlage bestimmter gespeicherter Daten und Algorithmen ein spezifisches Tag zuweist:

Beim Tagging wird jedes Token mit Informationen angereich[er]t. Die Art der Informationen kann sehr unterschiedlich sein. Ebenso vielfältig sind die Anwendungen, bei denen Tagging nützlich ist. Die Bezeichnung *Tag*, die mit *Etikett* übersetzt werden kann, deutet darauf hin, dass Tags sich immer auf genau ein Token beziehen. Der Aufbau tokenübergreifender Strukturen, wie z.B. beim Parsing, wird nicht unter Tagging zusammengefasst. Prinzipiell ist es aber möglich, Relationen zwischen Token mit Tags zu annotieren.

(<http://www.computing.dcu.ie/~jwagner/mag/node35.html>, Accessed 06. 08. 08)

Das Akronym 'POS' steht für *part of speech*, also die traditionelle englische Bezeichnung für 'Wortart' bzw. 'lexikalische Kategorie'. Ein POS-Tagger ist demnach ein Programm, welches jedes Token einer Eingabekette durch ein Tag annotiert, das Information über die Wortart des Tokens enthält.

Genau dieses ist in Abb. 1. auf der ersten Seite auch in Schritt 2 der lexikalischen Analyse, sprich der Klassifikation der Kette *The boy sleeps*, mit Bezug auf die jeweilige Wortart der Token, vorgenommen worden: The/Det | boy/N | sleeps/V.

Auch hier gilt, wie bei der Tokenisierung, dass das Verfahren dem unbedarften Betrachter ganz einfach erscheint, bei genauem Hinsehen jedoch schnell klar wird, dass es eine überaus komplexe Angelegenheit

ist. Grund: in den meisten Fällen ist es sehr schwer, eine derart klare Zuweisung zu treffen, wie im Beispielsatz *The boy slept*. Das wiederum hat damit zu tun, dass in der modernen Linguistik und der Computerlinguistik z.T. erheblich unterschiedliche Kategorieninventare verwendet werden.

**Wortarten in der Sprachwissenschaft**

Die Klassifikation von Wörtern ist im Rahmen der modernen Linguistik ein heißes, spannendes Eisen, auf dessen faszinierende Problematik hier nicht eingehen können. Stattdessen werden wir kurz ein paar unterschiedliche Modelle ansehen, um einige unterschiedliche Herangehensweisen an das Konstrukt 'Wortart' zu illustrieren.

Allgemein gilt, dass man Wortarten, also Klassen von Wörtern, auf Grundlage unterschiedlicher Eigenschaften der Wortform bilden kann. Grob gesagt spielen hierfür drei Kriterien eine Rolle:

- semantische Faktoren
- morphologische Faktoren
- syntaktische Faktoren

Eine ganz klassische (und unvollständige) Definition für Adjektive im Deutschen könnte so aussehen:

Prototypische Adjektive drücken Eigenschaften aus, werden nach Genus, Kasus und Numerus flektiert, können gesteigert werden und stehen zwischen einem Determinator und einem Nomen oder nach einer Kopula.

Hier finden wir semantische Faktoren ('drücken Eigenschaften aus'), morphologische Faktoren ('werden nach Genus, Kasus und Numerus flektiert und können gesteigert werden') und syntaktische Faktoren ('stehen zwischen Determinator und Nomen oder nach einer Kopula'). Auf ein Lexem wie KLEIN treffen alle die o.a. Faktoren zu.

Doch diese Definition ist unzulänglich, da in ihr nicht klargestellt wird, auf welcher Grundlage nicht-prototypische Wortformen zur Klasse gezählt werden können: TOT beispielsweise kann nicht gesteigert werden; ANGBLICH, welches ausschließlich prädikativ verwendet werden kann, steht entsprechend nie zwischen einem Determinator und einem Nomen. Trotzdem werden beide zu den Adjektiven gerechnet.

**Wortarten in der traditionellen Grammatik**

Auf Grundlage dieser Überlegungen ist es nicht weiter verwunderlich, dass auch im Rahmen der traditionellen bzw. Schulgrammatik unterschiedliche Kategorieninventare postuliert werden:

Nomen	Verb	Adjektiv	Artikel	Pro-nomen	Adverb	Konjunk-tion	Präpo-sition	Numerale	Inter-jektion
<i>Haus</i>	<i>liegen</i>	<i>rot</i>	<i>der</i>	<i>er</i>	<i>sehr</i>	<i>dass</i>	<i>auf</i>	<i>fünf</i>	<i>ach</i>
<i>Idee</i>	<i>sein</i>	<i>doppelt</i>	<i>ein</i>	<i>das</i>	<i>reichlich</i>	<i>oder</i>	<i>unter</i>	<i>dritte</i>	<i>hey</i>
<i>Rechner</i>	<i>schlafen</i>	<i>schnell</i>	<i>dies</i>	<i>es</i>	<i>schön</i>	<i>und</i>	<i>wegen</i>	<i>wenige</i>	<i>hoppla</i>
<i>Liebe</i>	<i>labern</i>	<i>schön</i>	<i>jenes</i>	usw.	<i>ziemlich</i>	<i>bevor</i>	<i>mit</i>	<i>Dutzend</i>	<i>Hallo</i>
<i>Tanz</i>	<i>geben</i>	<i>rund</i>	usw.		usw.	<i>seit</i>	<i>nach</i>	usw.	usw.
usw.	usw.	usw.				usw.	usw.		

**Abb. 17: 'Traditionelle Zehn-Wortarten-Lehre' (Quelle: Studienbuch Linguistik, Linke et al):**

Verb	Nomen	Adjektiv	Artikel, Pronomen	Adverb	Konjunk-tion	Präpo-sition	Inter-jektion
------	-------	----------	-------------------	--------	--------------	--------------	---------------

**Abb. 18: Wortarten im Duden Universalwörterbuch**

Wenn diese Inventare verglichen werden, fällt auf, dass Artikel und Pronomen in der Dudenklassifikation zusammenfallen und die Klasse 'Numerale' ganz verschwunden ist. Diese Klasse ist u.a. deshalb problematisch, weil die Gemeinsamkeit der Wörter in ihr ausschließlich inhaltlich geprägt ist: sie haben allesamt mit Quantitäten zu tun.

Intern kann diese Klasse aber durchaus noch unterteilt werden, z.B. in

- zwei, drei, vier... (Kardinal- oder Grundzahlen)
- zweite, dritte, vierte...(Ordinalzahlen)
- zweimal, dreimal, viermal...(Wiederholungszahlwörter, Iterativa)
- Hälfte, Drittel, Viertel...(Bruchzahlen)
- verdoppeln, verdreifachen...(Vervielfältigungszahlwörter, Multiplikativa)

Allerdings können wir *Hälfte*, *Drittel* usw. auch zu den Nomina zählen; *verdoppeln*, *verdreifachen* etc. zu den Verben, *zweimal*, *dreimal* etc. zu den Adverbem.

Was die Kardinalzahlen angeht, sind sich zwei aktuelle Wörterbücher des Deutschen, nämlich das bereits genannte Duden Universalwörterbuch und Wahrigs *Deutsches Wörterbuch* auch nicht einig: Duden führt diese Adjektive auf – Wahrig nicht.

**Wortarten in der modernen theoretischen Linguistik**

Wie bereits angesprochen wurde, ist die Wortartenproblematik ein viel diskutiertes Thema in der modernen Linguistik. Je nach Ansatz, theoretischem Hintergrund und Zielsetzung können unterschiedliche Klassen und Klasseninventare ausgemacht werden. Ein Problem beim Vergleich dieser Ansätze ist darin zu sehen, dass es den Vertretern häufig nicht, wie z.B. dem Duden, darum geht, eine vollständige Beschreibung einer Einzelsprache abzugeben, sondern stattdessen oft nur fragmentarisch bestimmte Sprachausschnitte und -daten behandelt werden; oft auch mit völlig unterschiedlichen Erkenntnisinteressen.

Dabei arbeiten durchaus viele moderne Grammatikformalisten mit Etiketten wie N (für Nomen), V (für Verb), A (für Adjektiv) usw. Dies suggeriert eine gewisse Traditionsbewusstheit, da diese Begriffe ja aus der traditionellen Grammatik übernommen wurden, und ebenfalls eine Vergleichbarkeit.

Dieser Eindruck täuscht allerdings. Stattdessen finden wir komplexe Modelle, in denen z.B. Hierarchien von Klassen repräsentiert sind, die strengen formalen bzw. formallogischen Bedingungen genügen müssen.

So werden z.B. die Etiketten 'N' und 'V' im Rahmen einer der Modellvarianten moderner **generativer Grammatik** chomskyscher Prägung als Attribute binärer Merkmale interpretiert, deren Kombination mit entsprechender Werteverteilung die folgenden Klassen liefert:

$$1. \begin{bmatrix} +N \\ -V \end{bmatrix} \quad 2. \begin{bmatrix} -N \\ +V \end{bmatrix} \quad 3. \begin{bmatrix} +N \\ +V \end{bmatrix} \quad 4. \begin{bmatrix} -N \\ -V \end{bmatrix}$$

**Abb. 19: Wortarten als Bündel binärer Merkmale**

Diesen Merkmalsbündeln könnten wir wie folgt die traditionellen Wortarten zuordnen: 1: Nomina, 2: Verben, 3: Adjektive 4: Partikel. Das ermöglicht es, Aussagen wie z.B. die folgenden über das Deutsche zu treffen:

- Diejenigen Klassen, deren Attribut N positiv spezifiziert ist, werden bezüglich Genus flektiert (trifft zu auf 1 (Nomina) und 3 (Adjektive))
- Diejenigen Klassen, deren Attribut V positiv spezifiziert ist, können prädikativ verwendet werden (trifft zu auf 2 (Verben) und 3 (Adjektive, die im Deutschen über die Kopula vermittelt werden)).

Wenn wir die Präpositionen unter Partikel zählen, können wir z.B. für das Englische eine Aussage treffen wie

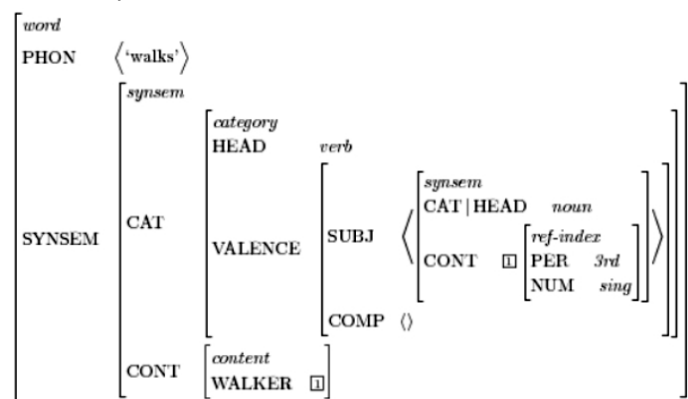
- Nur diejenigen Klassen, deren Attribut N negativ spezifiziert ist, weisen Kasus zu (trifft zu auf 2 (Verben) und 4 (Partikel, hier: Präpositionen)).

Die Verwendung von Merkmalen ist aus der modernen Linguistik nicht mehr wegzudenken. Einer der insbesondere in der Computerlinguistik führenden Ansätze, die **HPSG** (*Head-driven Phrase Structure Grammar*), ist komplett auf elaborierten Merkmalsstrukturen und -hierarchien aufgebaut und benötigt aufgrund seines ausgeprägten Lexikalismus *de facto* nur noch sehr wenige Regeln, um die Kombinatorik natürlicher Sprache abzubilden.

In diesem Grammatikmodell sind Etikette wie *Nomen* oder *Verb* Namen für Typen von komplexen Merkmalsstrukturen, die verschiedene Wortformen einer Einzelsprache abbilden. Die Merkmalsmatrix zeigt auf der nächsten Seite zeigt die HPSG-Struktur für das Wort *walks*.

Das Attribut-Wert-Paar [CAT: [HEAD: *verb*]] verweist dabei auf einen (abstrakten) Merkmalstyp namens *verb*, der die für Verben relevanten Merkmale (wie z.B. die Kongruenzmerkmale Numerus und Person oder ein Tempusmerkmal) enthält.

Das Merkmal mit dem Attribut *Valence* gibt an, wieviele und welche Argumente das Verb zu sich nimmt - im vorliegenden Beispiel, bei dem intransitiven *walks*, genau ein Argument, nämlich das Subjekt, welches seinerseits wieder durch eine komplexe Merkmalsstruktur beschrieben ist.



**Abb. 20: Lexikoneintrag in der HPSG**

Ein weiterer in der Computerlinguistik verbreitete Ansatz, die **Kategorialgrammatik**, ähnelt mit Bezug auf seine stark lexikalistische Grundausrichtung der HPSG. Grundelemente dieses Modells sind – wie es der Name schon sagt – die Kategorien, die sich durch ihre Form und die in ihnen enkodierte Information ganz erheblich von den lexikalischen und syntaktischen Kategorien wie z.B. N, VP, PP, A usw. in einer herkömmlichen Phrasenstrukturgrammatik unterscheiden.

In einer Kategorialgrammatik wird versucht, die syntaktische und die semantische Information, die mit einem Lexem oder einer Konstituente verknüpft ist, bereits in der Kategorie, die diesem Lexem oder dieser Konstituente zugeordnet ist, festzuhalten.

Einem intransitiven Verb wie *sleeps* beispielsweise würde in der Kategorialgrammatik eine syntaktische Kategorie zugeordnet, aus der klar hervorgeht, dass sich dieser Ausdruck zusammen mit einem NP-artigen Ausdruck zu einem Satz verbindet. Ohne auf die genauen Algorithmen einzugehen, die für eine detaillierte Darstellung nötig wären, betrachten wir die folgenden Lexikoneinträge und die damit erzeugbaren Strukturbeschreibungen:

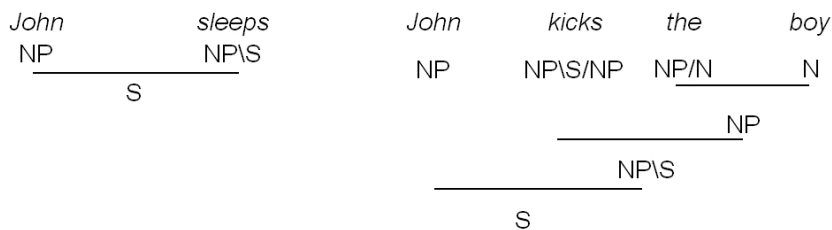
**Funktionsanwendung** (für eine KG mit bidirektionalem Operator):

Wenn  $x$  von der Kategorie  $A/B$  ist und  $y$  von der Kategorie  $B$  dann ist  $x y$  von der Kategorie  $A$   
 Wenn  $x$  von der Kategorie  $A/B$  ist und  $y$  von der Kategorie  $B$  dann ist  $y x$  von der Kategorie  $A$

**Lexikon:**

<i>John</i>	NP
<i>sleeps</i>	NP\S
<i>kicks</i>	NP\S/NP
<i>boy</i>	N
<i>the</i>	NP/N

**Ableitungen:**



**Abb. 21: Kategorien und Strukturbeschreibungen in der Kategorialgrammatik**

Eine Kategorie wie z.B. NP\S ist dabei im Sinne einer Funktion zu verstehen, deren Funktor S eine NP-Argument zu sich nimmt. Bei der Abarbeitung einer Eingabekette wie *John sleeps* würde auf der linken Seite dieser Funktion nach einer NP 'gesucht', wenn diese gefunden wird, kann NP\S per Funktionsanwendung zu S gekürzt werden.

In diesen Ausführungen geht es nicht darum, die genauen Interna der einzelnen Ansätze nachzuvollziehen, sondern darum, die Vielfältigkeit und Komplexität der Frage nach den lexikalischen Kategorien und dem Lexikon allgemein in der theoretischen Linguistik zu illustrieren.

Was die meisten Ansätze gemein haben, ist der bereits mehrfach erwähnte Lexikalismus: die mit einem Wort im Lexikon verbundene Information reicht weit über eine schlichte Zuweisung zu den traditionellen Klassen hinaus, in den meisten Modellen (ich kenne keines, bei dem es anders wäre) ist das Lexikon stark mit morphologischer, semantischer und syntaktischer Information angereichert.

**Wortarten in getaggtten Corpora**

Es gibt eine Vielzahl unterschiedlicher Corpora, d.h. elektronisch aufgearbeiteter Textsammlungen. Diese sind teilweise POS-getaggt, also mit Information über die Wortartzugehörigkeit der einzelnen Tokens angereichert. Die für diese Klassifizierung verwendeten Wortarten unterscheiden sich in (a) ihrer Bezeichnung (b) ihrer Anzahl und (c) der durch sie abgebildeten Information zum Teil erheblich von den traditionellen Wortarten und auch den in modernen, theoretischen Ansätzen verwendeten Kategorien.

Die in einem getaggtten Corpus verwendete Menge von Kategorien wird *Tagset* genannt. Das Tagset des getaggtten Brown Corpus umfasst ca. 90 Tags, das Penn-Treebank Tagset ca. 45, das Lancaster URCEL Tagset C5 ca. 60 Tags, das URCEL Tagset C7 ca. 145.

Eine Miniauswahl des C7 Tagsets sieht so aus:

AT	article	VVD	past tense of lexical verb
AT1	singular article	VVG	-ing participle of lexical verb
JJ	general adjective	VVI	infinitive
JJR	general comparative adjective	VVZ	-s form of lexical verb
JJT	general superlative adjective		



Was bei diesen Beispielen sehr deutlich wird, ist, dass die Klassen im wesentlichen auf Grundlage morphologischer Faktoren gebildet werden. So können wir eine Art abstrakte 'Oberklasse' V ausmachen, die auf der Basis verschiedener Verbformen in die Klassen VVD, VVG, VVI und VVZ unterteilt ist.

Im Grunde entsprechen diese Klassen dem, was im Rahmen der theoretischen Linguistik, genauer der generativen Grammatik, unter dem Stichwort 'Subkategorisierung' bekannt und angewendet wurde, bevor merkmalsbasierte Ansätze ihren Durchmarsch feierten: es wurden Lexika mit zahllosen Unterklassen verwendet; ein Verfahren, das bestimmte Probleme wie z.B. Kongruenzphänomene gut löste, die Grammatik aber ungeheuer aufgebläht hat.

Was in den Tagset-Kategorien jedoch fehlt, ist syntaktische Information: wiewohl wir Klassen wie 'Verbform Infinitiv', 'Verbform 3. Pers. Sg. Präsens' etc. finden, ist keine Klasse 'transitives Verb' oder 'intransitives Verb' auszumachen: die Wörter *sleeps*, *kicks* und *gives* z.B. gehörten alle zur Klasse VVZ.

Das wiederum bedeutet, dass eine syntaktische Weiterverarbeitung entsprechend getaggtter Einkabeketten ohne weitere Aufarbeitung oder Annotierung nicht mit einer implementierten Grammatik funktionieren würde, die auf entsprechende Information in ihrer Regelkomponente angewiesen ist.

## POS-Tagging Algorithmen

Im letzten Abschnitt dieses Textes soll es um die Frage gehen, wie denn nun genau ein Tagger verfahren kann, wenn es darum geht, eine tokenisierte Eingabe zu taggen.

Es ist relativ klar, dass ein Tagger irgendwie auf ein Tokenlexikon zurückgreifen muss, um seine Aufgabe erfüllen zu können. Wenn dem Tagger ein vollausgebautes Vollformenlexikon zur Verfügung steht, würde das Tagging nicht mehr bedeuten als einen Look-up des Tokens in eben diesem Lexikon und dann die Zuordnung der im Lexikon für das Token aufgeführten Kategorie. Wenn ein Token nicht gefunden wird, erfolgt ggf. eine Nachedition 'per Hand'.

### Grundlage: das Tagger-Training

Vielen Taggern aber steht kein solches Vollformenlexikon zur Verfügung. Sie werden stattdessen, um eine Grundlage für das Taggen zu bekommen, per eines bereits existierenden, getaggtten Corpus 'trainiert'.

Unter 'Trainer' versteht man in diesem Kontext einen Datensatz getaggtter Token, mithilfe dessen der zu trainierende Tagger über statistische Verfahren bestimmte Zuweisungsfrequenzen von Wörtern und Kategorien 'lernt'.

Nehmen wir dafür ein Beispiel. Die folgende Abbildung zeigt, wie man mit Python auf der Grundlage des im NLTK enthaltenen Brown-Korpus einen Datensatz erzeugt, der dann als Trainingsdatensatz für verschiedene Tagger zur Verfügung gestellt werden kann:

```
>>> import nltk
>>> from nltk.corpus import brown
>>> brown_trainer = brown.tagged_sents(categories='k')[122:124]
>>> print brown_trainer
[[('She', 'PPS'), ('wants', 'VBZ'), ('to', 'TO'), ('pay', 'VB'), ('you', 'PPO'),
 ('a', 'AT'), ('visit', 'NN'), ('.', '.')], [('She', 'PPS'), ('says', 'VBZ'),
 ('the', 'AT'), ('children', 'NNS'), ('miss', 'VB'), ('you', 'PPO'), ('.', '.')]]
```

Abb. 22 : Ein Mini-Trainingsset aus Daten des Brown Corpus

Nachdem die entsprechenden Module (`nltk` sowie `brown` aus `nltk.corpus`) importiert wurden, kommt die entscheidende Anweisung. Es wird ein Objekt namens `brown_trainer` erzeugt, welches aus Datensätzen des getaggtten Brown-Korpus besteht. Der Parameter `categories = 'k'` bedeutet, dass hier nur Sätze aus dem Datensatztyp 'General Fiction' vertreten sind. Der Index `[122:124]` bezieht sich auf die (für diese Illustration wahllos gewählten) Datensätze. Für einen 'echten' Trainer würden natürlich sehr viel mehr Daten benötigt, z.B. *alle* Daten aus dem getaggtten Brown-Korpus. Die Daten im getaggtten Brown Korpus sind Listen von Tupeln von Zeichenketten der Form `(['wort', 'kategorie'])`.

Wie aber kann ein Tagger mit derartigen Daten 'trainiert' werden? Nun, zum einen lernt er für alle im Korpus befindlichen Wörter die zugewiesene Kategorie. Über den oben angegebene Datensatz würde der Tagger z.B. 'lernen', dass das Token *she* der Kategorie PPS zuzuweisen ist, das Token *wants* der Kategorie VBZ, das Token *you* der Kategorie PPO usw. usf. Mit anderen Worten: der Tagger erzeugt sich über den Trainingsdatensatz sein eigenes Tokenlexikon. Wird er dann auf ein ungetaggttes Korpus angewendet, kann er die entsprechenden Zuweisungen auf Grundlage seines frisch erlernten Lexikons treffen.

Zum anderen entwickelt er auch eine Statistik darüber, wie oft welche Zuweisungen getroffen wurden, mit welcher Häufigkeit welche Tokens im Korpus zu finden sind und auch, welche Abfolgen von Tokens zu beobachten sind. Was den letztgenannten Punkt angeht, sind in diesem Kontext die sogenannten N-gramm-Tagger zu erwähnen.

### N-gramme und N-gramm Tagger

Das Konzept des N-grammes spielt im Rahmen der statistisch ausgerichteten Sprachtechnologie eine zentrale Rolle und wird z.B. bei der Erstellung von Konkordanzen<sup>7</sup> oder im Tagging eingesetzt.

Ein N-gramm ist dabei eine N-fache Folge von Zeichen. Ein Unigramm besteht aus einem einzelnen Zeichen, ein Bigramm aus zwei Zeichen, ein Trigramm aus drei Zeichen usw. Wenn mit Zeichen 'Buchstabe' gemeint ist, wäre A ein Unigramm, AA ein Bigramm, AAA ein Trigramm, AAAA ein Tetragramm, AAAAA ein Pentagramm usw.

Im Rahmen des Tagging kann sich 'Zeichen' sowohl auf 'Token' als auch auf 'Tag' beziehen. Nehmen wir als Beispiel zur Illustration die erste Liste des in Abb. 21 dargestellten Trainingssets:

```
[('She', 'PPS'), ('wants', 'VBZ'), ('to', 'TO'), ('pay', 'VB'), ('you', 'PPO'), ('a', 'AT'), ('visit', 'NN')]
```

**Abb. 23 : Ein Mini-Mini-Trainingsset aus Daten des Brown Corpus**

Aus diesem Datensatz können wir die folgende Kette von Tags bilden: PPS-VBZ-TO-VB-PPO-AT-NN.

In dieser Kette wiederum können wir z.B. die folgenden N-gramme ausmachen:

Bigramme: [PPS-VBZ], [VBZ-TO], [TO-VB], [VB-PPO], [PPO-AT], [AT-NN]

Trigramme: [PPS-VBZ-TO], [VBZ-TO-VB], [TO-VB-PPO], [VB-PPO-AT], [PPO-AT-NN]

Ein N-gramm Tagger nun ist ein Programm, das beim Durchlaufen eines Trainingskorpus auch Buch führt über die N-gramme, die in diesem Korpus zu finden sind. Ein Bigramm-Tagger würde eine Statistik über Bigramme anlegen, ein Trigramm-Tagger über Trigramme usw.

Das Mini-Mini-Trainingsset aus Abb. 22 würde von einem Bigramm-Tagger wie folgt statistisch aufbereitet:

Gesamtzahl Tokens:	7
Tokenfrequenzen:	she: 1, wants: 1, to: 1, pay: 1, you: 1, a: 1, visit: 1
Tagfrequenzen:	PPS: 1, VBZ: 1, TO: 1, VB: 1, PPO: 1, AT: 1, NN: 1
Token-Tag-Frequenzen:	she-PPS: 1, wants-VBZ: 1, to-TO: 1, pay-VB: 1, a-AT: 1, visit-NN: 1
Bigrammfrequenzen:	[PPS-VBZ]: 1, [VBZ-TO]: 1, [TO-VB]: 1, [VB-PPO]: 1, [PPO-AT]: 1, [AT-NN]: 1

**Abb. 24 : Statistik des Mini-Mini-Trainingssets aus Daten des Brown Corpus**

Welchen Zweck erfüllen derartige Statistiken? Um diese Frage zu beantworten, sehen wir uns ein kurzes, sehr konstruiertes Beispiel an. Das Beispiel ist deshalb konstruiert, weil auf diese Weise gewährleistet ist, dass die Datenanalyse noch mit bloßem Auge zu durchschauen ist. Es geht hier um das Prinzip, nicht die konkrete Anwendung – letztere soll ja anhand echter Daten im Rahmen des NLTK getestet werden.

### Wahrscheinlichkeitsberechnungen durch Tagger (informell)

Gegeben sei der Satz

*I saw the light.*

Bei den Wörtern in diesem Satz haben wir es in zwei Fällen mit kategorialer Mehrdeutigkeit zu tun, nämlich bei den Token *saw* und *light*. Uns kommt es nur auf *light* an, welches als Nomen, Adjektiv oder Verb klassifiziert werden könnte. Die Kernfrage, die sich für den Tagger stellt, lautet also: als was soll *light* getaggt werden?

Wir wollen, um eine Grundlage für die nachfolgenden Abschnitte zu bekommen, diese Kernfrage etwas anders postulieren, nämlich so:

Mit welcher Wahrscheinlichkeit ist *light* jeweils ein Nomen, ein Adjektiv oder ein Verb?

Grundlage für die Berechnung sind Erkenntnisse aus dem Bereich der Wahrscheinlichkeitstheorie. Dieses Teilgebiet der Mathematik entwickelt Modelle, die sich mit bestimmten zufälligen bzw. unvorhersehbaren Ereignissen beschäftigen und versuchen, eine Prognose über deren Ausgang zu berechnen. Für unser Beispiel ist nur die sog. bedingte Wahrscheinlichkeit relevant, welche wie folgt beschrieben werden kann:

Die bedingte Wahrscheinlichkeit ist die Wahrscheinlichkeit des Eintretens eines Ereignisses A unter der Bedingung, dass ein Ereignis B bereits vorher eingetreten ist.

Bedingte Wahrscheinlichkeit ist wie folgt formalisiert:  $P(A | B)$ , zu lesen als "wenn das Ereignis B eingetreten ist, ist die Wahrscheinlichkeit für das Eintreten von Ereignis A gegeben durch  $P(A | B)$ ".<sup>8</sup>

Diese Formel kann über die folgende Gleichung berechnet werden:

$$P(A | B) = \frac{P(A \cap B)}{P(B)}$$

In Worten: Die Wahrscheinlichkeit von A unter der Bedingung B ist die Wahrscheinlichkeit dafür, dass A und B gleichzeitig eintreten, geteilt durch die Wahrscheinlichkeit von B

<sup>7</sup> Eine Konkordanz ist ein Wörterbuch von Wörtern in einem Text inklusive der Kontexte, in denen diese Wörter auftreten

<sup>8</sup> P steht für Wahrscheinlichkeit (engl. *probability*), der senkrechte Strich bedeutet 'vorausgesetzt, dass...'

An dieser Stelle kann man schon sehen, wohin das Ganze im Falle eines Taggers führen wird. Gesetzt den Fall, wir hätten in einem getaggtten Korpus achtmal die Zuweisung *round:A* und dreimal die Zuweisung *round:N*. Insgesamt ist *round* 14 Mal im Korpus vertreten (z.B. auch noch als Adv und V). Die Kategorie A ist 26 Mal vertreten, die Kategorie N dagegen 38 Mal.

Wir können die Wahrscheinlichkeit dafür, dass *round* zu den Adjektiven bzw. den Nomina gezählt wird, nun wie folgt errechnen:

Ermittlung der Wahrscheinlichkeit dafür, dass Token<sub>x</sub> Tag<sub>y</sub> zugewiesen wird:

$P(\text{Token}_x|\text{Tag}_y)$ : Dividiere die Token-Tag-Frequenz von Token<sub>x</sub> durch die Tagfrequenz von Tag<sub>y</sub>

In unserem Beispiel wäre danach  $P(\textit{round}/A) = 8 : 26 = \mathbf{0,30}$  und  $P(\textit{round}/N) = 3 : 38 = \mathbf{0,08}$ . Ohne weitere Einschränkungen würde ein Unigramm-Tagger also das Token *round* als Adjektiv taggen.

Kommen wir mit diesen Erkenntnissen auf unser Beispiel zurück und also die Frage, mit welcher Wahrscheinlichkeit das Token *light* in dem Satz *I saw the light* als Nomen, Verb oder Adjektiv klassifiziert wird. Wir gehen von dem folgenden Korpus aus:

The light was burning brightly. A pale light crept through the curtains. We had a light meal for lunch. John switched the light off. A light breeze caressed her shoulders. He ought to light the candles.

Wird dieses Korpus per Hand getaggt, können wir aus den Daten die folgende Statistik ableiten (es sind nur die für uns relevanten Zahlen gegeben):

Gesamtzahl Tokens:	37				
Tokenfrequenzen:	light: 5	the: 4	a: 3		
Tagfrequenzen:	N: 9	A: 3	V:6	Det: 8	
Token-Tag-Frequenzen:	light-N: 3	light-A: 2	light-V: 1	the-Det:3	a-Det: 2
Bigrammfrequenzen:	[Det-N]: 6	[Det-A]: 3	[Det-V]: 0		

### Die Berechnung der Wahrscheinlichkeiten in einem Bigramm-Tagger

Die Wahrscheinlichkeiten werden über drei Schritte berechnet. Relevant sind die folgenden Elemente: Tokens wie *light*, Tags wie Det und Bigramme wie [Det-A].

Schritt 1: Ermittlung der Wahrscheinlichkeit dafür, dass Token<sub>x</sub> Tag<sub>y</sub> zugewiesen wurde:

$P(\text{Token}_x|\text{Tag}_y)$ : Dividiere die Token-Tag-Frequenz von Token<sub>x</sub> durch die Tagfrequenz von Tag<sub>y</sub>

Schritt 2: Ermittlung der Wahrscheinlichkeit dafür, Kat<sub>2</sub> in einem Bigrammes nach Kat<sub>1</sub> auftritt:

$P(\text{Kat}_2|\text{Kat}_1)$ : Dividiere die Bigrammfrequenz von [Kat<sub>1</sub>-Kat<sub>2</sub>] durch die Tagfrequenz von Kat<sub>1</sub>

Schritt 3: Bilde das Produkt aus  $P(\text{Token}_x|\text{Tag}_y)$  und  $P(\text{Kat}_2|\text{Kat}_1)$

Für unser Beispiel nun ergibt dieses Verfahren die folgenden Wahrscheinlichkeiten, aus denen sichtbar wird, dass der über das o.a. Korpus trainierte Bigramm-Tagger dem Token *light* in *I saw the light* das Tag 'A' zuordnen würde, da diese Zuordnung am wahrscheinlichsten ist:

Schritt 1:  $P(\textit{light}/N) = 3 : 9 = \mathbf{0,3}$        $P(\textit{light}/A) = 2 : 3 = \mathbf{0,7}$        $P(\textit{light}/V) = 1 : 6 = \mathbf{0,2}$

Schritt 2:  $P(N/\textit{Det}) : 6 : 8 = \mathbf{0,7}$        $P(A/\textit{Det}) : 3 : 8 = \mathbf{0,38}$        $P(V/\textit{Det}) : 0 : 8 = \mathbf{0}$

Schritt 3: für N:  $0,3 \times 0,7 = \mathbf{0,21}$       für A:  $0,7 \times 1 = \mathbf{0,27}$       für V:  $0,2 \times 0 = \mathbf{0}$

Dieses Resultat zeigt deutlich, dass für ein vernünftiges Taggen eben mehr und vor allem ausgewogenere Daten vorhanden sein müssen. Die 'falsche' Zuweisung von *light* zu A für den Beispieltext resultiert letztendlich darin, dass im Korpus zwar 9 N, aber nur 3 A vertreten sind. Schon ein einziges Adjektiv mehr im Korpus (das nicht durch *light* realisiert wäre) würde die Wahrscheinlichkeit der Zuweisung *light-A* auf 0,19 absenken.

Dennoch ist zu berücksichtigen, dass auch auf großen Korpora trainierte Tagger bis dato keine 100%-ig korrekten Ergebnisse erzielen, was bedeutet, dass für diejenigen Anwendungen, die auf derartig korrekte Ergebnisse angewiesen sind, i.d.R. ein Nacheditieren 'per Hand' erforderlich ist.