Gel speiende Spiegel und andere (linguistische) Probleme

Marcel Maci

Abstract

Ein Tool zur automatisierten Generierung von Stenogrammen zu schreiben ist eine schöne Aufgabe - eine ganz schöne sogar. Denn spätestens bei zusammengesetzten Wörtern zeigt sich, dass Lang- und Kurzschrift nicht einmal die sieben Buchstaben aus dem Wort "Schrift" gemein haben. Nebst Einzelwörtern komplizieren weitere Kinkerlitzchen wie Präfixe, Suffixe, Morphe und (unregelmässige) Stämme das Leben eines (germanischen) Linguisten auch ganz schön. Hier somit eine kurze Auslegeordnung der Einbahnstrassen, in die sich ein langschrift- und regelbasiertes Tool nichtsahnend (und zwangsläufig) verirrt - mit einigen Hinweisschildern und Bodenarbeiten, die helfen sollen, solche Sackgassen nicht oder zumindest nur so zu befahren, dass die Stenografie nicht auf der Strecke bleibt ...

Clever (aber) tricky kombiniert

Da wären wir also bei dem, was die Stenografie von der Langschrift unterscheidet: die Zeichen. Nur: Was ist ein Zeichen? In der Langschrift ist die Frage leicht zu beantworten: Zeichen = Buchstaben. Wobei Langschriftzeichen in etwa die phonetische Beschaffenheit eines Wortes wiedergeben sollen, aber gerade hier schon fangen die Probleme an: viel vs fiel, man vs Mann, mehr vs Meer. Man wird unschwer erkennen, dass in diesen Wörtern für die gleichen Phoneme (f, nn, langes e) nicht nur unterschiedliche, sondern auch eine unterschiedliche Zahl an Zeichen verwendet wird. Das wäre halb so wild, wenn sich aus der Schreibung wenigstens eine eindeutige Phonetik ableiten liesse, aber auch hier: Theater vs weither, Wien vs Ferien - sowohl th wie ie stehen hier für phonetisch komplett unterschiedeliche Lautanordnungen.

Von ihrer Seite her macht uns die Stenografie das Leben auch nicht leichter: Kurzschrift ist deshalb kurz, weil sie Dinge verkürzt und sie tut das mitunter, indem sie "Zeichen" (was immer das heisst, wie gesagt) fröhlich miteinander kombiniert. Beispiel: Glasträger vs Gastrecht vs Estrich. Stolze-Schreyaner/innen kombinieren in Gastrecht und Estrich s+t und hängen dann gleich noch ein r als Schlaufe ans gleiche Zeichen an; gewissermassen also drei (Langschriftzeichen) auf einen Streich. Glasträger hingegen wird mit separatem s (ein Zeichen) plus ein t (weiteres Zeichen) mit kombiniertem r geschrieben.

Aus den skizzierten Phänomenen lassen sich zwei Schlüsse ziehen: (1) jedes Übertragungstool (Langschrift zu Kurzschrift), das sich rein auf langschriftliche Buchstaben stützt, muss zwangsläufig scheitern und (2) den geschilderten Phänomenen (und damit richtigen Stenogrammen) kann man nur dann auf die Spur kommen, wenn man der schriftlichen Oberflächenstruktur der Wörter eine linguistische Analyse zur Seite stellt, die tiefer schürft und so die für die Stenografie relevanten Strukturen zu Tage fördert. Aber was für Strukturen sind das?

Zusammengesetzte Wörter

Da wären zuerst mal die zusammengesetzten Wörter: Gast|recht und Glas|träger trennen die Konsonantengruppe -str- an unterschiedlicher Stelle. Mit der Information | (Wortgrenze) lassen sich also zwei unterschiedliche Regeln schreiben: (R1) s|tr => [s][tr] und (R2) st|r => [st][r]. Aber wie finden wir - bzw. findet ein Computer - zusammengesetzte Wörter?

Eine Möglichkeit hierzu bietet die Umnutzung eines Spellcheckers. Wir schreiben wohlgemerkt: Umnutzung. Denn: Ein Spellchecker ist grundsätzlich dazu da, in einem Textverarbeitungsprogramm Rechtschreibfehler zu erkennen. Aber genau diese Eigenschaft lässt sich nutzen, indem man z.B. fragt, welche dieser linken und rechten Wortteile sind - für sich allein genommen - richtig geschrieben: Gas, Gast, Gastr vs Strecht, Trecht, Recht, echt. Der Spellchecker wird hier Gas, Gast, Recht und echt als Wörter erkennen. Daraus lässt sich eindeutig bestimmen, dass Gast und Recht die gesuchten Wörter sind.

Aber wie immer gibt es hier auch Spielverderber: Rohrohrzucker zum Beispiel.

Der Computer wird hier zum Schluss kommen, dass sowohl Roh|rohr|zucker als auch Rohr|ohr|zucker eine valable Antwort auf die Frage ist. Die Entscheidung, welche Variante nun die richtige ist, könnte hier nur semantisch erfolgen. Und um es gleich zu sagen: "Könnte" bedeutet, dass wir uns dieses Problem (Semantik) hier schon gar nicht um die Ohren schlagen wollen. Erstens gehört sie zu den schwierigsten Gebieten der Computerlinguistik. Zweitens werden wir mit unserer Methode doch - hoffentlich - wesentlich mehr richtige als falsche Resultate produzieren. Drittens haben wir tatsächlich noch ganz andere Probleme zu lösen. Zum Beispiel nur schon die Frage: Wie kommen wir zu den Wörtern, die wir zum Spellchecker schicken?

Die Bruteforce-Methode wäre vielleicht, jede mögliche Buchstabenkombination als potenziellen Wortteil in Betracht zu ziehen (also g, a, s, t, r, e, c, h, t; danach: ga, as, st etc.). Aber linguistisch macht das wenig Sinn: ein Wort besteht ja prinzipiell aus wenigstens einer Silbe.

Kombinatorische Silbentrennung

Silbentrennung ist ein brauchbares Mittel, um potenzielle Wörter zu finden. Unsere Annahmen: Jede einzelne Silbe oder eine Kombination mehreren Silben (in der vorgegebenen Reihenfolge) ist potenziell ein Einzelwort. Im Wort Glasträ-ger sind also folgende Möglichkeiten zu überprüfen: Glas, Trä, Ger (1 Silbe); Glasträ, Träger (2 Silben); Glasträger (3 Silben). Wenn n der Anzahl Silben entspricht, ergeben sich hier n * (n+1) / 2 Möglichkeiten. Sprich: Wir haben es hier mit einem Algorithmus zu tun, der (leider) exponentiell wächst. Da die deutsche Sprache gerne Wörter aneinanderreiht, bekommen wir das rechnerisch zu spüren: Wahr-schein-lich-keits-rech-nung (6 Silben) generiert zum Beispiel 21 Kombinationen.

Mit anderen Worten: Das Finden zusammengesetzter Wörter kann im Hinblick auf die Performance des Programmes zu einem Problem werden. Und zwar in verschiedener Hinsicht: Erstens konsultieren Spellchecker Wörterbücher und leiten mögliche Wortkombinationen zum Teil ihrerseits aus relativ komplizierten, internen Regeln ab (was an sich schon langsam ist). Zweitens wächst unser Problem wie gesagt exponentiell in Bezug auf die Silbenzahl. Und drittens wächst unser Problem prinzipiell linear in Bezug auf die Wortzahl: In einem längeren Text muss eine solche Analyse hintereinander einzeln für jedes Wort erfolgen - es sei denn, ein Teil der Wörter wiederholt sich und wir cachen sie (Ablage in einem Zwischenspeicher).

Selbstverständlich sind wir damit noch nicht am Ende unserer Sorgen. Andere Phänomene - wie zum Beispiel das Fugen-s komplizieren uns die Sache zusätzlich: Schifffahrts|kapitän. Der Spellchecker wird "Schiffahrts" kaltblütig als nicht korrekt geschriebenes (und somit inexistentes) Wort brandmarken. Wir können dies zwar umgehen, indem wir jede mögliche Kombination zusätzlich mit einem Bindestrich versehen. Dadurch verdoppelt sich aber die Anzahl der zu testenden Kombinationen: n * (n+1).

Ausserdem haben wir hier erst den ersten Teil der Arbeit erledigt: Wir wissen

nun, welche Teilwörter grundsätzlich existieren - aber nicht, welche Kombination von Teilwörtern tatsächlich eine gültige Abfolge ergibt.

Rekursiv durchs pyramidale Labyrinth

Die gefundenen, gültigen und (ungültigen) Teilwörter, lassen sich am besten als Pyramide nach Silben angeordnet darstellen.

1	Wahr Schein (Lich) (Keits) (Rech) (Nung)
2	(Wahrschein) (Scheinlich) (Lichkeits) (Keitsrech) Rechnung
3	Wahrscheinlich (Scheinlichkeits) (Lichkeitsrech) (Keitsrechnung)
4	Wahrscheinlichkeits (Scheinlichkeitsrech) (Lichkeitsrechnung)
5	(Wahrscheinlichkeitsrech) (Scheinlichkeitsrechnung)
6	Wahrscheinlichkeitsrechnung

Unsere Annahme ist nun, dass es eine Folge von (kleineren) gültigen Teilwörtern gibt, die in der Summe das zu analysierende Wort (letzte Zeile) ergeben. Da wir die kleinstmöglichen Bestandteile haben möchten, beginnen wir mit den einsilbigen Wörtern: wahr ist richtig, Schein auch, aber *Lich existiert nicht, also suchen wir auf der zweiten Zeilen (mit 2 Silben) weiter: *Lichkeits gibt es auch nicht, also versuchen wir es mit 3 Silben (*Lichkeitsrech) und schliesslich mit 4 (*Lichkeitsrechnung). Da dies nicht zum Ziel führt, ist Schein offensichtlich kein gültiges Teilwort und wir probieren hier eine Kombination mit 2 Silben (*Scheinlich) usw. usf.

Kurzum: Jedes Teilwort ist ein gültiges Wort, wenn es selber gültig ist und das folgende Teilwort ebenfalls ein gültiges Wort ist (oder kein weiteres Teilwort existiert). Auch hier haben wir es mit einem rekursiven Algorithmus zu tun, der der Performance nur weiter abträglich sein kann. Dafür ist der Algorithmus wirklich gründlich: Er findet wirklich die kleinsten möglichen Bestandteile, also zum Beispiel Dampf|schifffahrts|kapitäns|jacken|knopf|loch (6 Teile). Die menschliche, sprachliche Intuition würde hier vielleicht zusätzlich phrasieren: Dampf|schiffahrt(s) + Kapitäns|jacke + Knopf|loch (3 x 2 Teile). Aber auch hier: Solche Ordnungen entspringen wiederum semantischen Kriterien (und wir haben ja gesagt, dass wir davon die Finger lassen wollen:).

Hm ... Vorsilben?

Nun sind wir am Punkt, wo wir unseren Algorithmus über unseren Wortkorpus laufen lassen und stellen Erstaunliches fest: An|gabe, zu|geben, des|wegen, Ein|fall. So weit so gut: Der Computer hat also auch Präpositionen (an, zu) und Artikel (des, ein) als eigenständige Wörter erkannt. Sie machen als "Einzelwort in einem zusammengesetzten Wort" zwar wenig Sinn, aber wir können diese - so sagen wir uns - ja allenfalls dazu verwenden, Präfixe zu erkennen und im Hinblick auf gewisse stenografische Kürzungen zu nutzen.

Aber leider wird die Sache unschön und das aus zwei Gründen:

1. Es gibt eine Unzahl einsilbiger Wörter, die der Algorithmus nun querfeldein als Wortteile erkannt haben will und sie sind durchaus nicht nur als

Präfixe verwendbar: Au|tor, lachen|dem, an|genehm|er, Muse|um, Eulen|spie|gel.

2. Vorsilben, die wir im Hinblick auf stenografische Kürzungen benötigen würden, werden nicht als Präfixe (Einzelwörter) erkannt: Gedenken, gedeckt, Gelegenheit, gegangen.

Offenbar haben wir uns hier also ein neues Problem aufgehalst im Versuch, ein anderes loszuwerden. Denn: In lachen|dem würde der Computer nun auch lachen + Kürzung "dem" schreiben (anstatt das Zeichen -nd-) zu verwenden. Deshalb: Wie werden wir diese Geister, die wir gerufen haben, nun wieder los?

Think twice!

Unsere Knacknuss weist zum Glück ein erfreuliches Kriterium auf: Es sind kurze Wörter, ihre Anzahl also überschau- und in einer (mehr oder weniger kurzen) Liste erfassbar. Diese Liste können wir in einem Zwei-Schritt-Verfahren anwenden, um falsche Kombinationen herauszufiltern. Erster Schritt somit: Generieren aller möglicher Kombinationen. Zweiter Schritt: Anwenden von Filterregel (auf Liste basierend):

- den/dem/des/der am Wortende ist kein eigenes Wort
- au am Wortanfang ist kein selbständiges Wort (ausser vielleicht in Au|see)
- spie, gel, er sind in zusammengesetzten Wörtern in der Regel ebenfalls keine eigenständigen Wörter etc. (auch hier gibt es Ausnahmen wie Duschgel)

Ausserdem kommt uns vielleicht hier auch die Stenografie noch etwas zu Hilfe. Im System Stolze-Schrey werden die meisten zusammengesetzten Wörter verbunden geschrieben. Das heisst, es macht hier eigentlich keinen Unterschied, ob eine Wort- oder eine Silbengrenze erkannt wird. Wir können in den meisten Fällen also die (falschen) Wortgrenzen auch stehen lassen und sie als Synonyme für eine Silbengrenzen betrachten.

Präfixe?!

Aber eben: Was wir bis jetzt haben ist ein (mehr oder weniger zuverlässiger) Algorithmus zur Erkennung zusammengesetzter Wörter. Damit bewegen wir uns bezüglich gewissen stenografischen Phänomenen immer noch auf der (langschriftlichen) Oberfläche: Geben vs gemacht vs gegeben. Hier wird (in Stolze-Schrey) die Silbe ge- kurz geschrieben, wenn es sich um eine Vorsilbe (Präfix) handelt. Wie erkennen wir eine solche Vorsilbe?

Wie wir bereits gesehen haben, wird in Wörtern wie Ein|gabe oder An|gabe die Vorsilbe als eigenständiges Wort erkannt, weil sie (1) existieren (als Artikel oder Präposition) und (2) auch der zweite Wortteil (Gabe) existiert. Wir können den Fall ge nun ganz einfach auf die Fälle an und ein zurückführen, indem wir einfach zusätzlich definieren, dass ge ebenfalls als eigenständiges Wort behandelt werden soll. Dank der Bedingung, dass der zweite Teil ebenfalls ein

gültiges Wort sein muss, finden wir bei der rekursiven Suche die gewünschte Kombination: ge|macht (macht existiert), Ge|denken (denken existiert).

Im Anschluss machen wir uns den Zwei-Schritt-Ansatz zunutze, den wir bereits zum Ausfiltern falscher Wortteile verwendet haben zunutze. Anstatt die Wörter auszufiltern, können wir hier Regeln schreiben, die eine Wortgrenze in eine Morphemgrenze umschreiben, also zum Beispiel: zu| => zu+ und ge| => ge+ mit den Resultaten: zu|geben => zu+geben und ge|macht => ge+macht.

Präfix + Spellchecker + Suffix

Aber auch hier haben wir die Rechnung wieder ohne den Wirt gemacht, denn es gibt einmal mehr eine Liste von Miesepetern: Gelegenheit, gegangen. Hier existiert der zweite Teil des Wortes nicht (*legenheit, *gangen) und die Vorsilbe ge wird somit nicht erkannt. Auch hier können wir uns analog mit dem gleichen Prinzip weiterhelfen:

1. Wir definieren heit und en als zusätzlich gültige Wortteile
2. Die rekursive Suche findet nun: Ge|le|gen|heit oder Ge|leg|en|heit, ge|gang|en

Mit dem Zweischritt-Verfahren können wir nun Präfixe und Suffixe markieren: Ge+legen#heit, ge+gang#en. Beide Beispiele setzen natürlich voraus, dass im zweiten Schritt gleichzeitig gen und en im Wortinnern herausgefiltert werden. Bei -en ist dies relativ klar (kann nur als Endsilbe vorkommen) bei gen wäre es allenfalls durchaus möglich, dass Wortkombinationen existieren.

In beiden Fällen wurde für den Mittelteil - den wir auch als Stamm betrachten können - der Spellchecker verwendet. Es ist gut möglich, dass auch hier unregelmässige Stämme existieren, die wiederum nicht gefunden würden. Auch hier wäre die Möglichkeit, zusätzliche Wortteile als gültige Wörter (1. Schritt) zu definieren und am Schluss wieder herauszufiltern oder zu belassen (2. Schritt).

Kleinste Teilchen

Wörter im beschriebenen Verfahren in ihre kleineren und kleinsten (morphologisch "atomaren") Teile zu zerlegen hat seine Vorteile und Grenzen. Es fällt auf, dass mit jeder Verkleinerung der untersuchten Einheiten die Zahl der falschen Kombinationen zunimmt. Mit anderen Worten: Die Verfeinerung von Schritt 1 hat immer eine Vergrösserung des Aufwands in Schritt 2 zur Folge. Deshalb sollte das Verfahren abwägend angewendet werden: So kleine (und so wenige) Einheiten wie nötig (im Hinblick auf das Ziel, also z.B. die Umsetzung als Stenogramm), aber so gross wie möglich!

Spellchecker vs morphologische Analyse

Die obige Methode funktioniert soweit. Natürlich stellt sich die Frage, warum man nicht gleich auf ein Tool zurückgreift, das spezifisch für morphologische

Analyse ausgelegt ist (GERTWOL, SMOR, DEMorphy oder ähnliche). Die Frage ist berechtigt und wie bereits eingangs erwähnt: Mit einem Spellchecker (Rechtschreibeprogramm) nach Morphen zu suchen ist, als versuchte man eine Radmutter mit einem Flaschenöffner anzuziehen (es geht, ein bisschen, und dann wünschte man sich doch, man hätte einen Schraubenschlüssel).

Im Falle von VSTENO hat der Ansatz verschiedene Gründe:

- Viele Analysetools sind nur unter kommerzieller Lizenz verwendbar (und hier geht es nicht mal ums Geld, sondern um eine ideologische Allergie und Unverträglichkeit: VSTENO ist 100% freie Software).
- Wenn eine freie Lizenz (GPL oder MIT, wie im Falle von DEMorphy) verfügbar ist, dann stimmen unter Umständen andere Parameter nicht (DEMorphy, beispielsweise, ist in Python geschrieben; es wäre nicht unmöglich, DEMorphy von PHP aus zu verwenden, aber technisch dennoch nicht das naheliegendste).
- Ein sehr zuverlässiger und komplexer Spellchecker mit freier Lizenz (hunspell) wird auf jedem (Linux-)System bereits frei Haus mitgeliefert und kann ohne weiteres verwendet werden.

Als letztes kommt hinzu, dass nicht für jeden Einsatzzweck eine vollständige und übermässig tiefschürfende, morphologische Analyse nötig ist, da diese unter Umständen sogar ein regelrechter Overkill sein kann. Im Falle von Stolze-Schrey beschränken sich die relevanten morphologisch aufzuschlüsselnden Wortteile auf geschätzt ein Dutzend Vorsilben (ge, zu, dem, den, der, des, plus einige weitere).

VSTENO

VSTENO, in dessen Kontext die obigen Überlegungen formalisiert wurden, verwendet für seine linguistische Analyse lediglich zwei zusätzliche Tools: (1) php-Syllable (als Silbentrenner) und (2) hunspell (als Spellchecker). Beide Tools können verschiedene Sprachen verarbeiten, sodass prinzipiell auch anderssprachige Stenographie-Systeme umgesetzt werden können (die Sprache bzw. das Wörterbuch ist durch den/die Benutzer/in wählbar).

Diese beiden Tools, verbunden mit einer - ebenfalls durch den/die Nutzer/in definierbare - Liste an zusätzlichen Präfixen, Stämmen und Suffixen (zur Markierung von gültigen Wörtern/Wortteilen) und dem rekursiven Suchalgorithmus, ergeben einen sauberen Schritt 1 der linguistischen Analyse.

Die Nachbearbeitung (Schritt 2) ist dann für den/die Benutzter/in unter Verwendung der REGEX-Formelsprache frei und individuell programmierbar (als Analyzer-Regeln im Header-Teil der Model-Definition).

Wie bereits angedeutet ist eine linguistische Analyse per se rechnerisch einigermassen aufwändig. VSTENO bleibt hiervon nicht verschont und verliert spürbar an Ausführungsgeschwindkeit: Im Vergleich zu vorher (ohne linguistische Analyse) verläuft die Generierung der Stenogramme nun um den Faktor 2.5 langsamer (und stellt wesentlich höhere Ansprüche an die Rechenleistung und

den Speicher des Servers). Ausgedeutscht bedeutet eine Verlangsamung um den Faktor 2.5: Die linguistische Analyse verbraucht mehr Ressourcen als die eigentliche Berechnung der Stenogramme.

Es lohnt sich somit, zeitkritische Funktionen - wie zum Beispiel die rekursive Suche - nicht als PHP-Skript sondern als kompiliertes Modul zu verwirklichen[1]. Weitere Möglichkeiten der Geschwindigkeitsoptimierung sind das Caching (häufige Wörter[2])

[1] Diese Möglichkeit besteht zum Beispiel mit PHP-CPP (http://www.php-cpp.com/)

[2] Die häufigsten 207 Wörter der deutschen Sprache machen ca. 54% des Textes aus (https://de.wikipedia.org/wiki/Liste_der_h%C3%A4ufigsten_W%C3%B6rter_der_deutschen_Sprache). Besonders lohnenswert wäre es in diesem Fall aber, spezifische, längere Wörter zu cachen, die sich mehrmals wiederholen.