

Jana Pauls
Modulabschlussprüfung 3: Mediengeschichte
Institut für Medienwissenschaft
Philosophische Fakultät III
Humboldt-Universität zu Berlin
06.01.2014

Vom Medium der Sprache, der Schrift und des Digitalcomputers

*Ausführungen zur Sprachverarbeitung mit dem Computer
auf textlicher Ebene*

Inhalt

Einleitung	4
Teil 1: Das Programm	5
Aufgaben des Programms	5
Die Form des Limerick	5
Die Reime.....	5
Die Silben.....	6
Aufbau des Programms	7
1. Datenbank der Silben	7
2. Datenbank der Vokale	7
3. Grundvoraussetzungen/Hauptschleife.....	8
4. Generierung und Ausgabe des Verses.....	9
5. Determinierung des Reimes	9
6. Ermittlung möglicher Endsilben	10
7. Schluss der Schleife	10
Probleme und Restriktionen	12
Diphthonge	12
Mehrsilbige Reime	13
Phonem-Graphem-Korrespondenz.....	14
Teil 2: Sprache und Schrift – Medientheoretischer Bezug.....	16
Einleitung	16
Was ist Sprache, was ist Schrift?	16
Schriftspracherwerb an Mensch und Computer	17
Ist Schrift gleich Sprache? – Dependenz- und Autonomiehypothese	18
Anmerkung zur Relevanz.....	19

Fazit	20
Anhang	22
Programmlisting	22
Literaturverzeichnis	27

Einleitung

Die menschliche Sprache ist ein Phänomen für sich. Sie ist Ausdruck von Intelligenz, unterscheidet uns von anderen Wesen auf diesem Planeten. Die Arbeit im Feld der künstlichen Intelligenz hängt dadurch stark zusammen mit dem Versuch, Computern Sprache beizubringen.

Der Gegenstand dieser Seminararbeit ist nun die Frage, wie Computer Sprache verarbeiten, insbesondere wie Phänomene der gesprochenen Sprache in einem digitalen System repräsentiert werden, sowie der Zusammenhang zwischen den Medien Schrift und Sprache und seine Bedeutung für die Sprachfähigkeit von Computern. Als Ausgangspunkt für diese Überlegungen steht dabei ein in Locomotive Basic auf der Emulation eines Schneider CPC Heimcomputers selbst geschriebenes Programm, welches die Beispielaufgabe bearbeitet, aus einer festen Datenbank von Silben nonsemantische Limericks zu generieren.

Teil 1: Das Programm

Aufgaben des Programms

Die Aufgabe des Programms ist es, aus einer Datenbank vorgegebener Silben ein Gedicht der Form des Limericks, zu generieren. Die Semantik wird hierbei außen vor gelassen: Das fertige Gedicht soll keinen Sinn ergeben, es soll sich lediglich in Form, Rhythmus und Reim an die Vorgaben des Limericks halten.

Die Form des Limerick

Wolfgang Arndt beschreibt diese in seinem Buch *Deutsche Verslehre – Ein Abriss*: „Die fünf Zeilen des Limericks sind nach diesem Reimschema gegliedert: a a b b a. Die erste, zweite und letzte Zeile (mit dem Reim a) haben drei Hebungen, die dritte und vierte (mit dem Reim b) nur zwei. Auch die Anzahl und Lage der Senkungen sind durch das metrische Schema vorgegeben. In der Regel liegen zwischen den Hebungen zwei Senkungssilben. Jede Zeile beginnt mit einem ein- oder zweisilbigen Auftakt.“ (Arndt 1995, S. 170)

Um das Programm zu vereinfachen, habe ich mich auf nur eines der möglichen Versmaße eines Limericks festgelegt: Zwei Verse im dreihebigen Anapäst (je neun Silben), gefolgt von zwei Versen im zweihebigen Anapäst (je sechs Silben) und einem weiteren dreihebigen Anapäst im fünften Vers. Da hierdurch die letzte Silbe jedes Verses betont ist, sind alle Reime einsilbige (männliche) Reime¹.

Die Reime

Die Reime stellen das Hauptproblem dieser Aufgabe dar. Nach Arndt verstehen wir unter Reim „den Gleichklang zweier oder mehrerer Lautgruppen von ihrem letzten betonten Vokal an. [...] Entscheidend ist immer der Klang, niemals das Schriftbild.“ (Arndt 1995, S. 86) Letzteres ist insofern problematisch, dass unser Computer eben nicht mit Klang arbeitet, sondern nur mit dem Schriftbild. Auf diese Problematik werde ich später zurückkommen; für die Zwecke meines einfachen Limerick-Programms werden wir sie jedoch zunächst ignorieren. Die deutsche Sprache hat den Vorteil (im Vergleich zum Beispiel zum Englischen), dass in vielen Fällen Schriftbild und Klang miteinander korrespondieren, sodass gleich klingende Wortteile tatsächlich auch gleich geschrieben werden. Auf diese Fälle stütze ich die Daten für das Programm und definiere unter diesen Umständen den Reim als die Übereinstimmung zweier oder mehrerer Buchstabenketten ab ihrem letzten Vokal.

¹ Mögliche Variationen dieses Schemas werden im Abschnitt „Probleme und Restriktionen“ besprochen

Die Silben

Warum das Programm mit Silben arbeitet, statt mit ganzen Wörtern oder einzelnen Buchstaben, ist leicht erklärt. Wörter ins Versmaß des Limericks einzupassen wäre deutlich schwieriger, da nicht nur ihre unterschiedlichen Längen beachtet werden müssten (was an sich schon eine größere Datenbank an Wörter erfordern würde, damit genug ins Versmaß passende Kombinationen gegeben wären), sondern zudem auch noch die Betonung ihrer Silben. Von vornherein mit einzelnen Silben zu arbeiten, liegt daher nahe.

Eine andere Option wäre, die Datenbank auf einzelne Buchstaben und damit auf – je nachdem, ob die Sonderfälle des deutschen Schriftsystems beachtet werden – 26 bis 30 Elemente zu beschränken und das Programm die Silben selbst zusammenstellen zu lassen. Dies würde zwar zu vielseitigeren Ausgaben bei weniger benötigten Daten führen, allerdings haben Silben eine schon recht komplexe Struktur, an die das Programm sich bei ihrer Erstellung halten müsste. Ketten von Konsonanten in Silbenansatz und -koda treten beispielsweise aufgrund der sogenannten Sonoritätshierarchie nur in bestimmten Reihenfolgen auf. Diese Regeln könnten in einem Programm zwar inkludiert werden, auch dies ist aber für unsere Zwecke unnötig kompliziert, weswegen ich mich darauf beschränkt habe, dem Programm eine Datenbank von Silben vorzugeben. Es sei angemerkt, dass die Auswahl der Silben willkürlich ist und das Programm prinzipiell auch mit jeder anderen Auswahl arbeiten könnte, ausgenommen Silben, die Diphthonge enthalten (die Gründe hierfür sind im Abschnitt „Probleme und Restriktionen“ erläutert).

Aufbau des Programms

Im Folgenden wird das Basic-Programm in Teilabschnitten dargestellt und erklärt. Das vollständige Programmlisting am Stück ist im Anhang an diese Arbeit zu finden.

1. Datenbank der Silben

```
100 dim s$(151)      : REM Dimensionierung der Variable auf Anzahl der Silben in Data
110 for i=1 to 150
120 read s$(i)        : REM Einlesen aller Silben nach s$(i)
130 next i
140 data ba           : REM Silben-Datenbank
150 data bla
160 data da
170 data fa
180 data ha
.
.
.
```

Zu Beginn des Programms wird eine Datenbank von Silben eingelesen. Die hier verwendete Datenbank besteht aus 150 offenen (mit einem Vokal endenden) und geschlossenen (mit einem Konsonanten endenden) Silben². Prinzipiell kann die Datenbank aus einer beliebigen Anzahl beliebiger Silben bestehen. Damit diese einfache Version des Programms mit ihnen arbeiten kann, müssen allerdings gewisse Voraussetzungen erfüllt sein (s. dazu Abschnitt „Probleme und Restriktionen“).

Die Silben werden als Elemente der Variable s\$(i) eingelesen, welche zuvor so dimensioniert wird, dass sie die Anzahl der Elemente in der Datenbank enthalten kann.

2. Datenbank der Vokale

```
2000 dim v$(6)       : REM Dimensionierung der Variable auf die Anzahl der Vokale in Data
2010 for k=1 to 5
2020 read v$(k)       : REM Einlesen der Vokale nach v$(k)
2030 next k
2040 data a           : REM Datenbank der Vokale
2050 data e
2060 data i
2070 data o
2080 data u
```

² Die gesamte verwendete Datenbank würde an dieser Stelle unnötig viel Platz in Anspruch nehmen. Selbstverständlich ist sie aber im vollständigen Programmlisting im Anhang an diese Arbeit enthalten.

Damit das Programm in den Silben Reime erkennen kann, müssen diese auf übereinstimmende Vokale untersucht werden (s. Eigenschaften eines Reimes). Nun besitzt der Computer aber keine Definition davon, was ein Vokal ist, weswegen auch hierfür zuerst eine Datenbank eingelesen wird, die alle (für uns bedeutsamen) Vokale in der Variable v\$(k) ablegt, um sie später in der Abgleichroutine einzusetzen.

3. Grundvoraussetzungen/Hauptschleife

2100 a=9	: REM Festlegen der Silbenzahl für Vers 1
2200 dim ends\$(20)	: REM Dimensionierung auf Anzahl möglicher Reimsilben
2300 :	
3000 for m=1 to 5	: REM Schleife für 5 Verse
3010 l=int(rnd(1)*l)+1	: REM Zufälliges l zwischen 1 und vorherigem Wert von l
3020 if m=3 then l=0:a=6	: REM Variablen setzen für Vers 3
3030 if m=5 then a=8	: REM Zählvariable setzen für Vers 5

Dieser Programmteil eröffnet die Schleife, in der der Hauptteil des Programms abläuft. Da für die fünf Verse des Limericks prinzipiell jeweils die gleichen Aktionen durchgeführt werden (zufällige Silbenauswahl, gefolgt von Bestimmung des Reims und Suche nach einer Endsilbe für den nächsten Vers), ist es möglich, das Programm sehr kompakt als einzelne FOR-NEXT-Schleife zu formulieren.

Die Variationen, die in den einzelnen Versen erforderlich sind, werden hier zu Beginn der Hauptschleife mithilfe von IF-Abfragen gegeben: Für bestimmte Versnummern wird die später verwendete Zählvariable a (Variable für die Anzahl der zufällig generierten Silben) angepasst. Außerdem wird hier zu Anfang der Schleife für die Variable l ein zufälliger (ganzer) Wert zwischen 1 und dem vorherigen Wert von l festgelegt. Im ersten Durchlauf der Schleife bleibt dieser Wert zunächst bedeutungslos; ab dem zweiten Durchlauf ist er dafür verantwortlich, dass aus allen dem Reim nach möglichen Endsilben eine zufällige für den zu generierenden Vers ausgewählt wird (s. Zeile ...). Deswegen wird l auch für den dritten Vers auf 0 zurückgesetzt, da dieser einen neuen Reim eröffnet und daher keine zuvor ermittelte Endsilbe benötigt wird.

Die Dimensionierung von ends\$(l) ist notwendig, weil in dieser Variable später alle Silben abgelegt werden, die einen bestimmten Reim enthalten. Je nach vorliegender Datenbank muss diese Variable also verschieden dimensioniert werden.

4. Generierung und Ausgabe des Verses

```

3050 for j=1 to a           : REM Schleife für die Generierung a zufälliger Silben
3060 i=int(rnd(1)*150)+1    : REM zufälliges i zwischen 1 und 150
3070 vers$(m)=vers$(m)+s$(i) : REM Anhängen der zufälligen Silbe an generierten Vers
3080 if j=3 or j=6 then vers$(m)=vers$(m)+" " : REM Leerzeichen nach je 3 Silben
3090 next j                 : REM nächste Silbe
3100 vers$(m)=vers$(m)+ends$(l) : REM zuvor ermittelte Endsilbe anhängen
3010 print vers$(m)         : REM Ausgabe des Verses

```

Diese Schleife generiert eine durch die Variable a bestimmte Anzahl zufälliger Silben aus s\$(i) und schreibt diese hintereinander in die Variable vers\$(m) (m ist die Nummer des Durchlaufes der Hauptschleife und so die Nummer des Verses), wobei aus rein ästhetischen und lesepraktischen Gründen hinter jeder dritten Silbe (hinter jedem vollständigen Anapäst also) ein Leerzeichen eingefügt wird. Zuletzt wird dem Vers (sofern dieser sich auf einen bereits bestehenden Vers reimen muss) die zuvor ermittelte Reimsilbe angehängt, bevor der gesamte Vers schließlich ausgegeben wird (in Vers 1 und 3 ist diese Reimsilbe leer).

5. Determinierung des Reimes

```

3130 for k=1 to 5           : REM Schleife durch alle 5 Vokale
3140 r=instr(s$(i), v$(k))   : REM Position des Vokals in der letzten Silbe
3150 reim$=right$(s$(i), len(s$(i))-(r-1)) : REM Reim der Silbe wird abgelegt
3160 if r=0 then next k      : REM wenn Vokal nicht enthalten,
                             : nächster Vokal

```

Zwei Silben reimen sich, sofern sie ab dem ersten Vokal in der Silbe übereinstimmen (sich also nur im Silbenkopf unterscheiden). Damit das Programm diesen Silbenreim feststellen kann, sucht es also mithilfe der Vokal-Datenbank in v\$(k) die Stelle des Vokals im zuletzt verwendeten String s\$(i), also in der letzten Silbe des vorangegangenen Verses. Hierfür wird jeder Vokal der Reihenfolge in der Datenbank nach ausprobiert, solange bis die zur Suche verwendete INSTR-Funktion einen Wert >0 ausgibt, ein Vokal also gefunden wurde. Die Tatsache, dass die im Alphabet später vorkommenden Vokale hierbei nicht ausprobiert werden, stellt insofern kein Problem dar, dass eine Silbe in der Regel nur einen Vokal enthält (Diphthonge stellen eine Ausnahme dar, s. hierzu Abschnitt „Probleme und Restriktionen“). Der Reim wird nun zur weiteren Verwendung in der Variable reim\$ abgelegt.

```
3180 if m=1 then reim1$=reim$
3190 if m=4 then reim$=reim1$
```

Diese beiden Zeilen stellen eine vielleicht etwas unelegante Lösung dafür dar, dass der letzte Vers im Limerick nicht den Reim des hervorgegangenen Verses aufgreifen muss, sondern den des ersten. Im ersten Durchlauf des Programms wird der Reim also an dieser Stelle in die Variable reim1\$ geschrieben, wo er im vierten Durchlauf (für den fünften Vers) wieder abgerufen wird.

6. Ermittlung möglicher Endsilben

```
3210 l=1          : REM Zählvariable l für Anzahl zurücksetzen
3230 for i=1 to 150 : REM Schleife durch alle verfügbaren Silben
3240 if instr(s$(i), reim$) <> 0 and instr(s$(i), reim$)+len(reim$)-1=len(s$(i)) then
ends$(l)=s$(i):l=l+1 : REM Wenn Reim am Ende der Silbe enthalten,
                     : ablegen in ends$(l)
3250 next i       : REM nächste Silbe überprüfen
```

Mit dieser Schleife werden alle Silben in der Datenbank daraufhin untersucht, ob sie den zuvor extrahierten Reim reim\$ enthalten. Alle Silben, bei denen dies der Fall ist, werden in die Variable ends\$(l) geschrieben, jedoch nur, wenn der Reim tatsächlich am Ende der Silbe steht (so wird verhindert, dass das Programm beispielsweise auf „du“ eine Silbe wie „um“ reimen würde, in der der ermittelte Reim „u“ ja im Prinzip auch enthalten ist). In einem Satz: Wenn der Reim in der zu untersuchenden Silbe enthalten ist, und seine Position addiert zu seiner Länge eine Stelle mehr ergibt, als die Silbe lang ist, der Reim also am Ende der Silbe steht, wird die Silbe als potenzielle Endsilbe des nächsten Verses in die Variable ends\$(l) abgelegt. Dies wird mit allen Silben der Datenbank wiederholt, sodass ein Pool sich reimender Silben in ends\$(l) entsteht.

7. Schluss der Schleife

```
3270 l=l-1        : REM verhindert leere Variable ends$(l) für das letzte l
3280 a=a-1        : REM nächster Vers hat eine Zufallssilbe weniger
3400 :
3500 next m       : REM Wiederholung der gesamten Schleife für den nächsten Vers
```

Die letzten Zeilen der Hauptschleife bereiten die benötigten Variablen für den nächsten Durchlauf vor. Von l wird 1 abgezogen, da für den höchsten Wert von l die Variable ends\$(l)

leer ist und nicht verwendet werden kann. a wird ebenfalls um 1 verringert, da für jeden Vers, der sich auf den vorherigen reimt, die letzte Silbe durch den Reim bestimmt und daher eine zufällige Silbe weniger benötigt wird. Alternativ hierzu könnte man auch im ersten Teil der Hauptschleife für jeden Vers die Anzahl zufälliger Silben einzeln festlegen.

Nach allen fünf Durchläufen der Programmschleife sieht die Ausgabe dann beispielsweise so aus:



```
senime kimmachlach wilermalt
kiversag saghikler maltkrakalt
tridaru hastramoll
trefurprim triglotoll
laprejag kridichtich malterfalt
Ready
█
```

Liest man die fünf Verse im richtigen Rhythmus (je zwei Silben unbetont, dann eine betont), ergibt sich vom Klang her ein Limerick.

Probleme und Restriktionen

In der einführenden Erklärung zu Aufgaben und Funktion des Programms wurden gewisse Restriktionen bereits erwähnt, die ich hier weiter ausführen will. So kann das Programm so, wie es hier aufgeführt ist, nur mit bestimmten Silben und Reimarten arbeiten und unterliegt Willkürlichkeiten in der Korrespondenz zwischen Schrift- und Klangbild in der deutschen Sprache. Erdenkliche Erweiterungen des Programms, die diese Beschränkungen verringern, werden hier dargestellt.

Diphthonge

Wie bereits angedeutet, ergibt sich aus dem Ansatz, Reime über das Enthaltensein des gleichen Vokals in der Silbe zu finden, das Problem, dass Diphthonge im Silbenkern zu falschen Zuordnungen führen. Da die Suchschleife zur Überprüfung des Vokals in der Silbe die Vokale in alphabetischer Reihenfolge ausprobiert und stoppt, sobald ein Vokal gefunden wurde, kann es bei Diphthongen (wie z.B. „ie“) vorkommen, dass nur die zweite Hälfte dessen als Silbenkern erkannt wird. Aus der Silbe „vier“ würde als Reim „er“ erkannt werden und nicht korrekt „ier“.

Nun sollte man meinen, dass zumindest Diphthonge, deren erster Vokal auch im Alphabet vor dem zweiten liegt, keine Probleme bereiten. Das stimmt insofern, dass bei diesen Diphthongen der Reim richtig extrahiert wird. In der Silbe „haus“ beispielsweise erkennt das Programm „a“ als Kernvokal der Silbe und extrahiert so den Reim „aus“. Unglücklicherweise treten allerdings Fehler auf, wenn eine Datenbank mit diphthongbasierten Silben auf einen Reim eines einzelnen Vokals durchsucht wird, der ein Substring der Diphthong-Silbe ist. Während auf „haus“ nur Silben, die in „aus“ enden, gereimt werden, würde das Programm kein Problem darin sehen, auch auf eine Silbe wie „bus“ Silben mit der Endung „aus“ zu reimen, da im Prinzip ja auch in ihnen der Reim „us“ in Endposition enthalten ist.

Das zuerst angesprochene Problem ist relativ einfach zu lösen. Soll der Suchalgorithmus auch auf diese Silbenkerne anwendbar sein, muss das Programm um eine weitere Suchschleife ergänzt werden, die nach der Abfrage auf einzelne Vokale die betreffende Silbe außerdem auf besagte Diphthonge untersucht und die Variable „reim\$“ gegebenenfalls überschreibt, nämlich sofern der INSTR-Wert des gefundenen Diphthongs kleiner ist als der des vorher gefundenen Vokals.

Das könnte in etwa so aussehen:

```
3130 for k=1 to 5
3140 r1=instr(s$(i), v$(k))
3150 reim$=right$(s$(i), len(s$(i))-(r1-1))
3160 if r=0 then next k
3161 :
3162 for k=6 to 21
3163 r2=instr(s$(i), v$(k))
3164 if r2<r1 and r2<0 then reim$=right$(s$(i), len(s$(i))-(r2-1))
3165 if r=0 or r2>r1 then next k
```

Das zweite angesprochene Problem präsentiert sich etwas schwieriger in seiner Lösung. Eine Option wäre, auch in dem Programmteil, der für das Suchen von auf den Reim passenden Silben verantwortlich ist, wieder eine Schleife einzubauen, die einzelne Vokale von Diphthongen unterscheidet. In die von mir gewählte Form des Programms ist eine solche Unterscheidung allerdings nicht auf sich mir erschließende Weise einzubauen.

Mehrsilbige Reime

Durch den von mir gewählten Rhythmus für den Limerick, enden die vom Programm erstellten Verse immer auf betonten Silben und haben so nur einsilbige, sogenannte männliche, Reime. Damit das Programm auch mit zweisilbigen (weiblichen) Reimen umgehen könnte, müsste es zuerst erkennen, ob dem Versmaß nach der Reim männlich oder weiblich sein muss. Bei einem strikten Versmaß wie dem hier gegebenen (welches sich immer noch an regelmäßige Anzahlen von Hebungen, Senkungen und Auftakten hält) ist das eine relativ einfache Aufgabe. Das Programm könnte nach dem Zufallsprinzip dem ersten Vers entweder neun oder zehn Silben zuschreiben; bei neun Silben ist der Reim männlich, bei zehn Silben weiblich. Der nächste Schritt wäre dann, im Falle eines weiblichen Reimes, die vorletzte, also die neunte, Silbe während des zufälligen Aufbaus des Verses beiseite zu nehmen und in eine Variable zu legen und die gesamte Routine zur Extraktion des Reimes mit dieser anstatt mit der letzten Silbe durchzuführen. Die letzte, zehnte Silbe müsste ebenfalls in einer Variable abgelegt werden, damit sie für den zweiten und fünften Vers jeweils angehängt werden könnte, um den zweisilbigen Reim zu vervollständigen.

Diese Erweiterung macht das Programm fähiger aber auch den Code deutlich länger, da sie nur sehr umständlich in der kompakten Schleifenstruktur unterzubringen ist und meiner

Vorstellung nach besser durch Subroutinen zu programmieren wäre, die sich jeweils mit den einzelnen Versformen befassen würden.

Phonem-Graphem-Korrespondenz

Unter Phonem-Graphem-Korrespondenz versteht man die Beziehung zwischen dem Klang- und Schriftbild der Sprache. Diese ist, wie zuvor angedeutet, nicht eindeutig oder in jedem Kontext gleich. Ein Phonem ist die kleinste bedeutungsunterscheidende Lauteinheit einer Sprache, also ein Laut, für den sich sagen lässt, dass es Wörter gibt, die sich von anderen nur in diesem einen Laut unterscheiden. (vgl. Dürscheid 2012, S. 130f.) In der deutschen Sprache gibt es 44 solcher Phoneme, jedoch nur 30 Buchstaben bzw. minimal mehr Grapheme³. (vgl. Weinhold 2005, nach Duden 1998) Es ist daher offensichtlich ausgeschlossen, dass jedes Graphem einem einzelnen Phonem entspricht, und genauso, dass jedes Phonem einem einzelnen Graphem entspricht: Der gleiche Laut kann durch verschiedene Grapheme repräsentiert werden, wie zum Beispiel /f/ in „Vogel“ und „Fliege“. Erziehungswissenschaftlerin Swantje Weinhold führt in ihrem Aufsatz „Schriftspracherwerb“ weitere Beispiele an: Ein Graphem steht kontextbedingt für verschiedene Laute, wie zum Beispiel das <e> in den Worten „Decke“, „Segel“, „Besen“ oder <ch> in „Sichel“, „Kachel“, „Christa“. Andere Buchstaben dienen als „Leseanweisung zur Dehnung und Schärfung von betonten Vokalen in prominenten Silben“, wie in Wörtern wie „Bohne“, „Waage“ oder „Ziegel“ (Weinhold 2005, S. 7).

Was also deutlich wird, ist, dass sich der Klang eines Wortes oder einer Silbe nur in wenigen Fällen tatsächlich deutlich durch das Schriftbild ausdrücken lässt bzw. nur vom Schriftbild nicht auf den Klang und nur vom Klang nicht aufs Schriftbild zu schließen ist. Für ein Computerprogramm, das auf textlicher Ebene Reime generieren soll, ist dies offensichtlich ein Problem. Zwar gilt, da auf dieser Ebene nicht mit ganzen Wörtern gearbeitet wird und Semantik unbeachtet bleibt, dass sich alle Silben, die parallele Graphemgruppen aufweisen, auch parallel auslauten lassen; das Programm gibt also keine falschen Reime aus. Allerdings ist seine Fähigkeit zu reimen eben doch auf ebendiese Reime beschränkt, die auch im Schriftbild übereinstimmen. Einfach erscheinende Reime wie „Boot“ und „Not“ oder „Wald“

³ Zur Unterscheidung von Buchstabe und Graphem: Ein Buchstabe stellt die schriftliche Realisierung der kleinsten Einheit von Sprache dar. Von einem Graphem spricht man, wenn es sich bei dieser Realisierung um eine bedeutungsunterscheidende Einheit handelt. So ist beispielsweise <sch> ein Graphem, welches aus den drei Buchstaben <s>, <c> und <h> besteht. (vgl. Dürscheid 2012, S. 130f.)

und „kalt“ (die letzteren beiden Silben sind in meiner Beispieldatenbank enthalten, werden vom Programm aber niemals im gleichen Reim verwendet werden) bleiben unerkannt.

Man könnte so weit gehen, zu sagen, dass die vom Computerprogramm generierten Reime tatsächlich keine Reime sind. Wenn wir den Reim wie zuvor als Gleichklang mehrerer Lautgruppen definieren, trifft dies auf die vom Programm erstellten Reime zwar zu, jedoch nur durch Zufall, da in unserem Szenario bestimmte Voraussetzungen herrschen. Ließen wir das Programm beispielsweise anstatt mit deutschen Silben mit englischen einsilbigen Wörtern arbeiten, kämen dabei auch Reime wie „heard“ (Vergangenheitsform von engl. „hören“) auf „beard“ (engl. „Bart“) heraus, die nur im Schriftbild aber eben nicht im Klang übereinstimmen. Das Programm arbeitet also ausschließlich mit Symbolen, nicht mit tatsächlicher Sprache. Genauso könnte es diese Reime des Schriftbilds nämlich zudem auch in Zeichenketten erkennen, die überhaupt nicht, oder nicht nur, aus Buchstaben bestehen. Zwei übereinstimmende Symbolketten aus Sternchen, Strichen und Punkten oder sonstigen Sonderzeichen sind aber wohl nach keiner Definition als Reim zu bezeichnen. So stellen sich die Fragen, woher Buchstaben diese Sonderstellung einwohnt, und was genau Schrift überhaupt mit Sprache zu tun hat. Diese Fragen sollen den Ausgangspunkt für meine weiteren Überlegungen darstellen.

Teil 2: Sprache und Schrift – Medientheoretischer Bezug

Einleitung

Im Vorherigen Abschnitt hat sich nach Betrachtung der Funktionsweise eines einfachen Programmes zur textbasierten Ermittlung von Reimen die Frage nach dem Verhältnis von Sprache und Schrift herauskristallisiert. Kann ein phonologisches Phänomen wie der Reim auch in Schriftform existieren? Inwiefern gehören Buchstaben, im Gegensatz zu anderen Symbolen, zur Sprache? Lässt sich von einem textbasierten Programm überhaupt sagen, dass es mit Sprache arbeitet? Auf diese Fragen sollen im Folgenden Antworten gefunden werden. Hierzu kläre ich zuerst, was wir unter Schrift verstehen und wie der Mensch sich ein Verständnis ihrer und des Zusammenhangs zwischen Lauten und Schriftbild aneignet. Danach übertrage ich dieses Konzept auf den Computer, um darzustellen, wie auch eine schriftbasierte Maschine mit Laut-Phänomenen wie dem Reim theoretisch kompetent umgehen kann. Zuletzt diskutiere ich Rolle des Mediums Schrift in Bezug auf das Medium Sprache und ihre Implikationen für die kommunikatorische Unabhängigkeit des Computers vom Menschen.

Was ist Sprache, was ist Schrift?

Wir haben bereits festgestellt, dass die erste und natürlichste Art, einem Computer eine sprachliche – oder tatsächlich irgendeine – Aufgabe zu stellen, über die Schrift läuft. Wenn wir einen Computer programmieren, geben wir ihm seine Anweisungen schriftlich. Dies hat seinen Grund in einer Grundeigenschaft von Schrift: Wie der Computer selbst⁴ ist auch sie gewissermaßen digital.

Schrift besteht im Gegensatz zu gesprochener Sprache aus diskreten Einheiten. (vgl. Dürscheid 2012, S. 32) Für den Computer, der auf unterster Ebene nur mit diskreten Werten – genau zwei an der Zahl: 1 und 0 bzw. an und aus – arbeiten kann, sind diese viel leichter zu berechnen und darzustellen als der kontinuierliche Strom an Signalen, den die gesprochene Sprache darstellt.

Die diskreten Einheiten, aus denen Schrift besteht, sind die Grapheme. Dass nicht jedes Graphem einem Laut entspricht und umgekehrt, wurde bereits erklärt und wird erneut

⁴ In dieser Arbeit wird grundsätzlich vom Computer im heute gängigen Sinne des Digitalcomputers, nicht des Analogrechners, ausgegangen.

deutlich, wenn man sich die Unterschiedlichkeit in der Struktur von Schrift und gesprochener Sprache vor Augen ruft. Grapheme sind dennoch die sichtbaren Repräsentanten der Laute unserer Sprache. Sie kodieren sozusagen die gesprochene Sprache und wie jeden Code müssen wir erst lernen sie zu lesen. Da dies bei den meisten von uns im Kindesalter geschieht, kommen uns unsere Schrift und ihr Bezug zur Sprache als etwas Selbstverständliches vor. Im Prinzip ist diese Beziehung aber arbiträr und nur durch Konvention existent. Was Ferdinand de Saussure bereits über das sprachliche Zeichen schrieb, gilt genauso für das schriftliche: „Das Band, welches das Bezeichnete mit der Bezeichnung verknüpft, ist beliebig; und da wir unter Zeichen das durch die assoziative Verbindung einer Bezeichnung mit einem Bezeichneten erzeugte Ganze verstehen, so können wir einfacher sagen: das sprachliche Zeichen ist beliebig.“ (de Saussure 1931) Analog zu Saussures Aussage, dass diese Beliebigkeit des sprachlichen Zeichens durch die Existenz verschiedener Sprachen und unterschiedlicher Begriffe für dasselbe Bezeichnete zu beweisen ist, beweist sich die Beliebigkeit des schriftlichen Zeichens im unterschiedlichen Phonemsatz verschiedener Sprachen, die (annähernd) dasselbe Schriftsystem verwenden.

Schriftspracherwerb an Mensch und Computer

Wenn ein Kind also Lesen lernt, wird ihm beigebracht bestimmte Grapheme mit bestimmten Lauten zu assoziieren. Da dies aufgrund der nicht eindeutigen Phonem-Graphem-Korrespondenz nicht ausreicht, lernt es mit der Zeit durch bestimmte Regeln und eigene Erfahrung, Grapheme in ihrem Kontext zu lesen und zuzuordnen, sie zu Silben und schließlich zu Wörtern zusammenzusetzen. (vgl. Weinhold 2005, S. 8f.) So erlernt also ein von gesprochener Sprache ausgehender Mensch den Zusammenhang zwischen Laut und Schrift.

Gehen wir nun vom schriftbasierten Computer aus, ergibt sich die Frage, ob man diesem nicht auf ähnliche Art „Lesen“ beibringen kann. Mit tatsächlichen Lauten kann dieser zwar nichts anfangen, jedoch kann man ihm gewiss das theoretische Wissen, zum Beispiel in Form von lautschriftlichen Beschreibungen, darüber vermitteln, welche Grapheme in welchem Kontext auf Silben- und Wortlevel welche Phoneme repräsentieren. Wie beim Kind, würde man wohl auch beim Computer damit anfangen, einzelnen geschriebenen Wörtern oder Silben eine phonemische Beschreibung zuzuweisen bzw. Regeln für die jeweilige Lautung der Grapheme in bestimmten Kontexten zur Verfügung zu stellen. Diese könnten durch Datenbanken mit

Beispielen von Wörtern gestützt sein (analog zum menschlichen Gedächtnis, in welchem bereits bekannte Wörter zusammen mit ihrer schriftlichen Repräsentation abgespeichert sind). Allerdings würde das nicht reichen, um eigenständiges Lesen zu konstituieren, da neue, unbekannte Wörter nicht korrekt erfasst werden könnten. Auch sämtliche Regeln für die Aussprache aller Wörter oder auch nur Silben per Hand zu programmieren wäre nicht ideal. Wie ein Kind, muss auch der Computer eigene Erfahrungen machen, um die Korrespondenz von Sprache und Schrift in seiner Ganzheit zu erfassen. Hierfür benötigt er lediglich große Mengen von Daten und die richtige Programmierung, um aus diesen durch logische Schlüsse selbst Regeln zu formen. Während dieses Vorhaben die Möglichkeiten einer mit Basic programmierten Emulation des Amstrad CPC wohl übersteigt, gibt es in der modernen natürlichen Sprachverarbeitung Modelle, sogenanntes unüberwachtes Lernen, nach denen Computer in einer gegebenen Menge von Daten Muster finden und daraus selbstständig lernen. (vgl. Ghahramani 2004)

Ist Schrift gleich Sprache? – Dependenz- und Autonomiehypothese

Wie wir sehen, ist es also durchaus möglich einem Computer ein Verständnis vom Klangbild einer Sprache, zusätzlich zum Schriftbild, beizubringen, und damit auch ein Programm zu entwerfen, welches in einer gegebenen Sprache verlässlich reimt. Eines ist allerdings zu beachten: Selbst wenn der Computer einen Reim korrekt generiert und diese auf dem Bildschirm ausgegeben hat, ist dies noch kein Reim im eigentlichen Sinne. Solange nur die Schriftform existiert, handelt es sich um eine symbolische Repräsentation des Reimes, nicht um einen tatsächlichen Reim. Bedeutet dies, dass auch in anderen Instanzen der Computer niemals wirklich mit Sprache arbeitet, sondern immer lediglich mit Repräsentationen? Ja und nein.

Die Besonderheit des Phänomens Reim ist, dass dieser sich durch seine Klanglichkeit definiert. Das heißt, er existiert nur in der gesprochenen Sprache; bis er nicht ausgesprochen wird, ist ein Reim kein Reim. Ob Schrift aber im Allgemeinen nur eine Repräsentation von gesprochener Sprache darstellt, oder aber eine Realisationsform von Sprache in sich selbst, darüber besteht Uneinigkeit unter Linguisten. Es bestehen zwei gegensätzliche Hypothesen, die von Christa Dürscheid (2012, S. 35ff.) ausgeführt werden:

Die Anhänger der sogenannten Dependenz-Hypothese argumentieren, dass Schrift die Sprache lediglich visualisiert, also Grapheme nur den bereits existenten Phonemen zugeordnet

wurden, da Sprache sowohl kulturgeschichtlich als auch in der Entwicklung des Individuums früher auftritt als Schrift. Sprache kann ohne Schrift existieren, Schrift ist ohne Sprache jedoch unvorstellbar und daher von dieser abhängig. Außerdem wird darauf hingewiesen, dass gesprochene Sprache öfter und in unterschiedlicheren Situationen zum Einsatz kommt als die geschriebene.

Die Autonomiehypothese hingegen sieht „die Schrift als eigenständige Realisationsform von Sprache“. Anhänger dieser Position argumentieren mit der strukturellen Verschiedenheit von Schrift und gesprochener Sprache (diskret vs. kontinuierlich) und ihrer teils sehr getrennten Funktion als speicherbare Form von Sprache, sowie mit der Beobachtung, dass Lesen und Schreiben sich nicht immer auf die gesprochene Sprache beziehen: „Das Geschriebene wird vom geübten Leser ganzheitlich erfasst, nicht Buchstabe für Buchstabe dekodiert. [... Es] erfolgt also nur selten der Umweg über die gesprochene Sprache.“ (Dürscheid 2012, S. 38) Zudem werden Bereiche der Sprache wie z.B. Wortbildung in einigen Fällen durch die Schrift beeinflusst. Akronyme bzw. Initialwörter etwa, die durch die Aneinanderreihung (und Aussprache) der jeweiligen Anfangsbuchstaben eines aus mehreren Elementen bestehenden Begriffs zu dessen Abkürzung gebildet werden, stellen einen solchen Fall dar.

Anmerkung zur Relevanz

Die Frage, ob Schrift als eigenständiger Teil der Sprache oder als bloßer Repräsentant dieser aufgefasst wird, ist insofern relevant, dass von ihr die Antwort auf die Frage abhängt, ob Computer menschliche Sprache verarbeiten können. Letztendlich arbeitet jeder (digitale) Computer mit diskreten Einheiten, also immer mit einer Verschriftlichung der gesprochenen Sprache, sei es nun Schrift im Rahmen des Alphabets oder des binären Nummernsystems. Nach der Autonomiehypothese handelt es sich auch hierbei um eine reelle Form von Sprache; betrachtet man das Problem allerdings aus der Perspektive der Dependenztheoretiker, bleibt der Computer in seinen kommunikatorischen Fähigkeiten abhängig von der Interpretation durch den Menschen.

Fazit

Wir haben gesehen, dass sich gewisse sprachliche Aufgaben schon mithilfe sehr simpler Programme vom Computer lösen lassen. Das größte Problem stellt hierbei die Korrespondenz zwischen der lautlichen und schriftlichen Ebene der Sprache dar. Prinzipiell ist es dem Computer möglich, das Verhältnis dieser Ebenen auf ähnliche Weise verstehen zu lernen wie der Mensch es tut. Im Gegensatz zum Menschen ist die Maschine allerdings aufgrund ihrer digitalen Funktionsweise immer an das Medium der Schrift gebunden. Ob die Schrift selbst eine Realisationsform von Sprache ist oder diese lediglich symbolisch repräsentiert, ist hierbei Ansichtssache, und so bleibt auch die Frage ungeklärt, ob Computer Sprache unabhängig vom Menschen verarbeiten.

Anhang

Programmlisting

```
100 dim s$(151)
110 for i=1 to 150
120 read s$(i)
130 next i
140 data ba
150 data bla
160 data da
170 data fa
180 data ha
190 data ja
200 data kra
210 data la
220 data ma
230 data pa
240 data sta
250 data stra
260 data tra
270 data ya
280 data za
290 data be
300 data bre
310 data de
320 data fle
330 data ge
340 data he
350 data me
360 data ne
370 data pre
380 data re
390 data se
400 data sche
410 data te
420 data tre
430 data ve
440 data bi
450 data ci
460 data di
470 data fi
480 data hi
490 data ki
500 data kri
510 data li
520 data mi
530 data ni
```

540 data pi
550 data pri
560 data sti
570 data tri
580 data wi
590 data bro
600 data co
610 data dro
620 data flo
630 data go
640 data glo
650 data ho
660 data kro
670 data lo
680 data mo
690 data no
700 data ro
710 data so
720 data scho
730 data wo
740 data du
750 data dru
760 data fu
770 data gru
780 data hu
790 data klu
800 data lu
810 data nu
820 data plu
830 data ru
840 data stu
850 data schu
860 data tu
870 data yu
880 data zu
890 data ab
900 data fab
910 data hab
920 data kab
930 data stab
940 data tab
950 data ald
960 data fald
970 data wald
980 data alt
990 data falt
1000 data kalt
1010 data malt

1020 data am
1030 data bam
1040 data dam
1050 data jam
1060 data sam
1070 data tram
1080 data ag
1090 data frag
1100 data jag
1110 data pag
1120 data sag
1130 data tag
1140 data ach
1150 data fach
1160 data dach
1170 data lach
1180 data krach
1190 data mach
1200 data en
1210 data fen
1220 data nen
1230 data sten
1240 data sen
1250 data ten
1260 data ven
1270 data el
1280 data bel
1290 data mel
1300 data pel
1310 data sel
1320 data tel
1330 data er
1340 data ber
1350 data kler
1360 data ler
1370 data scher
1380 data ver
1390 data ich
1400 data dich
1410 data gich
1420 data sich
1430 data tich
1440 data im
1450 data dim
1460 data kim
1470 data prim
1480 data wim
1490 data oll


```
1500 data boll
1510 data foll
1520 data toll
1530 data moll
1540 data um
1550 data bum
1560 data drum
1570 data kum
1580 data sum
1590 data ur
1600 data fur
1610 data lur
1620 data nur
1630 data tur
1640 :
2000 dim v$(6)
2010 for k=1 to 5
2020 read v$(k)
2030 next k
2040 data a
2050 data e
2060 data i
2070 data o
2080 data u
2090 :
2100 a=9
2200 dim ends$(20)
2300 :
3000 for m=1 to 5
3010 l=int(rnd(1)*l)+1
3020 if m=3 then l=0:a=6
3030 if m=5 then a=8
3040 :
3050 for j=1 to a
3060 i=int(rnd(1)*150)+1
3070 vers$(m)=vers$(m)+s$(i)
3080 if j=3 or j=6 then vers$(m)=vers$(m)+" "
3090 next j
3100 vers$(m)=vers$(m)+ends$(l)
3110 print vers$(m)
3120 :
3130 for k=1 to 5
3140 r=instr(s$(i), v$(k))
3150 reim$=right$(s$(i), len(s$(i))-(r-1))
3160 if r=0 then next k
3170 :
3180 if m=1 then reim1$=reim$
3190 if m=4 then reim$=reim1$
```

```
3200 :  
3210 l=1  
3230 for i=1 to 150  
3240 if instr(s$(i), reim$) <> 0 and instr(s$(i), reim$)+len(reim$)-1=len(s$(i)) then  
ends$(l)=s$(i):l=l+1  
3250 next i  
3260 :  
3270 l=l-1  
3280 a=a-1  
3290 :  
3300 next m  
3400 :  
3500 end
```

Literaturverzeichnis

- Arndt, Erwin. *Deutsche Verslehre : ein Abriß*. Berlin, Volk und Wissen, 1996.
- Dürscheid, Christa. *Einführung in die Schriftlinguistik*. Göttingen, Vandenhoeck & Ruprecht, 2012.
- de Saussure, Ferdinand. Hrsg. Bally, Charles. *Grundfragen der allgemeinen Sprachwissenschaft. Cours de linguistique générale*. Berlin, de Gruyter, 2001.
- Weinhold, Swantje. „Schriftspracherwerb“. In: Lange, G.; Weinhold, S. (Hrsg.), *Grundlagen der Deutschdidaktik. Sprachdidaktik, Mediendidaktik, Literaturdidaktik*. Baltmannsweiler, Schneider Verlag Hohengehren, 2005. S. 2-33.
- Ghahramani, Zoubin. „Unsupervised Learning“. 2004.
URL: <http://mlg.eng.cam.ac.uk/zoubin/papers/ul.pdf> [Abrufdatum 03.01.2014]