4 Suchen in symbolischer Information

4.1 Einführung

Unter symbolischer Information verstehen wir im engeren Sinne Daten, die sich auf Signale beziehen. Als einfachstes Beispiel könnte man eine Verschriftung einer Aufnahme als symbolische Information bezeichnen. Man nennt solche Informationen 'symbolisch' (manchmal auch 'kategorial'), um sie von den physikalischen Signalen zu unterscheiden. Wir können folgende Tabelle aufmachen, die den Unterschied etwas verdeutlicht:

	Physikalisches Signal	Symbolische Information
Repräsentation	Abtastwerte (Zahlen)	Zeichenketten
Speicherbedarf	Hoch	Niedrig
Informationstyp	Empirisch	Kategorial (meistens)
Beispiele	Sprachsignal	Transliteration
	Laryngo-Signal	Segmentierung
	F0-Verlauf	Akzente
	Formantverläufe	Formantpositionen/-breiten

Andere Beispiele für symbolische Information:

Transliteration:	Liste der Wörter plus Marker für andere Ereignisse, z.B. Geräusche, linguistische Effekte, Hesitationen, Pausen, Zögerungen, etc.
Transkript:	Liste der Phoneme bzw. Allophone einer Äußerung
Segmentierung:	Liste von Teilstücken einer Äußerung, den Segmenten
Phonetische Segmentierung:	Liste der Allophone mit segmentaler Information
Silbensegmentierung:	dito mit Silben statt Allophonen
Wortsegmentierung:	dito mit Wortgrenzen
Dialogakt-Segmentierung:	Abbildung der Wortkette auf *Dialogakte*
Prosodische Segmentierung:	Position und Klasse des prosodischen Ereignisses
Syntaxbaum:	Syntaktische Struktur einer Äußerung

Als *Meta-Informationen* bezeichnen wir zusätzliche wichtige Informationen zu einer Äußerung, die sich aber nicht direkt auf das Signal beziehen: Sprecher: Name, Geschlecht, Alter, Herkunft, Ausbildung, Größe, Raucher/Nichtraucher, Dialekt, Dialekt der Eltern, Grundschule, Beruf, Wohnort Aufnahmebedingungen: Mikrophon, Aufnahmetechnik, Raum, Akustik, Geräuschquellen, Task, Art der Sprache (spontan/gelesen etc)

Für den Phonetiker, der mit sehr großen Datenmengen experimentiert, ist es essentiell, solche symbolischen Informationen automatisch verarbeiten zu können. Z.B. um bestimmte Klassen von Signalen aus einem Korpus zu sammeln. Daher lernen wir im Folgenden einige wichtige UNIX-Tools zur Manipulation von symbolischer Information kennen.

4.2 Beispiele für symbolische Daten

Kopieren Sie die Dateien (mit allen Unterverzeichnissen!) in `/homes/ekurs2/SS10/Symbolic-Data` in Ihr Arbeitsverzeichnis und versuchen Sie deren Art einzuordnen in:

- Sprecher-Information
- Aufnahmebedingungen
- Symbolische Information zu einer Aufnahme

```
w026_pk.trl
w026_pk.trp
g121a/*.par
w026_pk.rpr
ABD.spr
```

4.3 Der Suchbefehl grep

Der UNIX-Befehl `grep` dient zum Suchen *in* Dateien (i.G. zum Befehl `find` oder `locate`, die zum Suchen *von* Dateien dienen!)

Im Directory `g121a` liegt eine Sammlung von sog. Partitur-Files (Endung *.par). Eine solche Sammlung nennen wir (zusammen mit den Signalen!) *Korpus*. Wir wollen zunächst feststellen, wie oft die Hesitation `'<"ah>'` in diesem Korpus vorkommen. Der Befehl `grep` sucht zeilenweise in Dateien nach bestimmten Mustern. Der allgemeine Aufruf ist dabei:

```
cip1 % grep <Muster> file1 [file2 file3 ...]
```

Alle Zeilen, in denen das Muster vorkommt, werden zusammen mit dem Filenamen ausgegeben. Z.B.

```
cip1 % grep '<"ah>' g121a/*.par
```

gibt alle Zeilen aus, in denen der String `'<"ah>'` vorkommt.

Nun fällt sofort auf, daß jedes `'<"ah>'` mehrfach vorkommt, weil dieses Label (eine MArkierung) nicht nur in einer *Spur* (einer Beschreibungsebene) vorkommt. Wir können das vermeiden, indem wir uns vorher mit einem weiteren 'grep' auf die Zeilen einer einzigen Spur beschränken, z.B. auf die Spur 'ORT' (was eine 'Spur' genau ist, wird im nächsten Abschnitt ausführlich beschrieben):

```
cip1 % grep '^ORT:.*<"ah>' g121a/*.par
```

Das ist der gleiche Befehl wie oben, aber das Muster ist komplizierter geworden. Es enthält sog. *Meta-Zeichen* mit besonderer Bedeutung, wie sie in *Regulären Ausdrücken* (Mustern) vorkommen.

4.4 Reguläre Ausdrücke

Ein regulärer Ausdruck ist ein *Muster* für eine Gruppe von Strings. Dazu werden sogenannte Metazeichen verwendet, die nicht das Zeichen bedeuten, sondern eine besondere Bedeutung haben:

`^`	: Zeilenanfang
`$`	: Zeilenende
`.`	: beliebiges Zeichen
`*`	: beliebige Wiederhohlung des vorangegangenen Zeichen (auch 0-mal!)
`[...]`	: genau ein Zeichen der Menge ...
`[^...]`	: genau ein Zeichen NICHT aus der Menge ...

`'^ORT:.*<"ah>'` bedeutet also im Klartext:
'Suche die Zeilen, die am Zeilenanfang mit 'ORT:' beginnen, dann eine beliebig lange Kette von beliebigen Zeichen (auch 0) und dann den String `'<"ah>'` enthalten.'

(Die einfachen Hochkommata vor und nach dem regulären Ausdruck sind notwendig, weil hier im Ausdruck ein `'"'` und ein `'*'` vorkommt. Würde man die Hochkommata weglassen, würde die Shell meinen, man gibt ihr einen Ausdruck mit Joker (*) und versuchen, dazu passende Filenamen zu finden.)

Nun haben wir alle Zeilen auf dem Bildschirm, aber eigentlich wollten wir nur wissen, wieviele es sind. Also zählen wir in einer Pipeline einfach die Zeilen:

```
cip1 % grep '^ORT:.*<"ah>' g121a/*.par | wc -l
```

Zählen Sie auf die gleiche Weise auch die Hesitationen `'<"ahm>'`, `'<hm>'` und `'<h"as>'` und addieren Sie alle zusammen. Dann zählen wir, wieviele Äußerungen der Korpus überhaupt enthält:

```
cip1 % ls g121a/*.par | wc -l
```

Wenn wir die beiden Zahlen untereinander dividieren, bekommen wir die mittlere Anzahl von Hesitationen `'<"ah>'` pro Äußerung in diesem Korpus.

4.5 BAS Partitur Format (BPF) : Einführung

4.5.1 Was ist eine Partitur?

In der Phonetik ist das physikalische Signal fast immer ein Sprachsignal. Zu dem Signal kann es aber eine praktisch unbegrenzte Anzahl verschiedener symbolischer Beschreibungsformen geben. Um diese geordnet bearbeiten und darstellen zu können, stellen wir uns vor, dass diese in einer Art Musik-Partitur als verschiedene Stimmen über das gleiche Grundthema angeordnet sind. Daher der Name *BAS Partitur Format*.

Die einzelnen Stimmen der der Partitur nennen wir *Spuren* oder englisch *tiers* (gebräuchlich sind auch *layers*, *levels*, usf.). Alle haben irgendeinen Bezug zum zugrundeliegenden Sprachsignal, können also zeitlich *synchronisiert* werden. (Genau wie auch die Noten einer Partitur synchron übereinander angeordnet sind!). Darüber hinaus gibt es aber noch weitere Möglichkeiten, die einzelnen Tiers zu verbinden, und zwar über sog. Wort-Links oder *symbolische Links*. Ein symbolischer Link ist ganz einfach eine Zahl, die uns sagt, welchem 'Wort' das beschriebene Ereignis zugeordnet werden soll. Mit diesen beiden einfachen Mechanismen können nun alle möglichen Arten von symbolischer Information geordnet beschrieben werden.

(Skizze : Beispiel einer Partitur)

4.5.2 Wie sieht eine solche Partitur-Repräsentation als Datei aus?

Möglichst einfach. Ein *Partiturfile* besteht aus einer 7bit ASCII-Datei, die auf jedem Rechner zu lesen und zu bearbeiten ist (Plattform-Unabhängigkeit). Jede Zeile beginnt mit einem *Label*, daß uns sagt, welche Information aus dieser Zeile gelesen werden kann (man sagt auch: jedes Label definiert eine Syntax und eine Semantik der folgenden Zeile). Zum Beispiel

```
SPN: KAP
```

sagt uns diese Zeile, dass der Sprecher das Kürzel `'KAP'` hatte. Die Label sind natürlich einheitlich festgelegt und können in der Dokumentation [1] nachgeschaut werden.

[1] www.phonetik.uni-muenchen.de/Bas/BasFormatsdeu.html

Der erste Teil des Files ist ein genormter *Header*, d.h. ein Teil der uns allgemeine Informationen (Meta-Daten) über das Sprachsignal gibt. Z.B.

- Name des zugehörigen Signalfiles
- Abtastrate
- Anzahl der Bytes pro Abtastwert
- Bitauflösung
- Byteorder (01 oder 10)

Aber auch Dinge wie:

- Name oder Kürzel des Sprechers
- Datum der Aufnahme
- Name des Korpus etc.

Der Header-Teil beginnt immer mit dem Label `'LHD'` und ended mit dem Label `'LBD'`.

Beispiel für einen einfachen Header:

```
LHD: Partitur 1.2.3
REP: Muenchen
SNB: 2
SAM: 16000
SBF: 01
SSB: 16
NCH: 1
SPN: KAP
LBD:
```

Die erste Zeile (LHD) sagt uns die verwendete Version des Partitur Formats. Die Zeile `'REP'` definiert den 'place of recording', in diesem Falle 'Muenchen'. Die Zeile `'SNB'` sagt uns, daß pro Abtastwert 2 Bytes verwendet wurden. Die Zeile `'SAM'` definiert die 'sampling rate', die Abtastrate, zu 16kHz. Die Zeile `'SBF'` gibt die Reihenfolge der Bytes an: 01 Die Zeile `'SSB'` gibt die Bitauflösung wieder, hier 16 Bit. `'NCH'` schließlich ist die Anzahl der in diesem File gespeicherten Kanäle (1) und `'LBD'` schliesst den Header-Teil ab.

Nach diesem Header-Teil, der ein Minimum an Informationen enthalten muß, folgen beliebig viele *Spur-Blöcke*, die die einzelnen Tiers der Partitur beschreiben. Wieder hat jeder dieser Blöcke sein eigenenes Label und seine eigene Syntax und Semantik. Die Reihenfolge der einzelnen Blöcke spielt keine Rolle.

Beispiel:

Die folgenden Zeilen definieren eine sog. `ORT` Spur (orthographische Spur), d.h. die nackten gesprochenen Wörter:

```
ORT: 0  tsch"u"s
ORT: 1  denn
```

Diese Äußerung enthält also nur zwei Wörter. Wörter sind im Partitur Format so definiert, dass sie alle semantischen Einheiten bezeichnen, die vom Artikulationstrakt des Sprechers stammen. Geräusche ohne semantischen Gehalt (wie Atmen, Husten,etc.) werden nicht als Wörter gezählt. Ein Grenzfall sind Häsitationen wie 'ähm' oder 'hm'; diese werden als Wörter gezählt. Wie man sieht, enthält dieser Block überhaupt keine Informationen darüber, wo denn nun die beiden Worter im physikalischen Sprachsignal liegen. Es handelt sich hier um einen 'Klasse 1' tier, das ist ein tier, der nur rein kategoriale Elemente enthält, ohne irgendeinen direkten Bezug zur Zeitachse. Andere Klasse 1 Tiers sind z.B.:

Transliteration

```
TR2: 0 <#Klicken> tsch"u"s <#Klicken>
TR2: 1 denn .
```

Kanonische Aussprache

```
KAN: -1 <nib>
KAN: 0  tS'y:s
KAN: -1 <nib>
KAN: 1  dEn
```

Dialogakte

```
DAS: 0,1 @(BYE BA)
```

In allen Klasse 1 tier steht in der zweiten Spalte der symbolische Link, der die Zuordnung der Worteinheiten regelt. In allen vier Beispielen (die aus einen echten Partitur File stammen) ist der '0' das Wort `'tsch"u"s'` und der '1' das Wort `'denn'` zugeordnet. Im letzten Beispiel (Dialogakte) sieht man eine weitere Möglichkeit, nämlich daß eine Zeile sich auf mehrere Wörter bezieht. Das Label `'@(BYE BA)'` (Dialogakt Verabschiedung) bezieht sich auf '0,1', also auf eine Wortfolge, die aus Wort '0' und Wort '1' besteht.

Was für andere Klassen gibt es?

Es gibt insgesamt nur 5 verschiedene Grundklassen. Klasse 1 haben wir oben bereits kennengelernt. Die anderen 4 Klassen sind:

Klasse 2	: Spuren mit zeitlicher Relation, zeitkonsumierend
Klasse 3	: Spuren mit zeitlicher Relation, nicht zeitkonsumierend
Klasse 4 zeitkonsumierend	: Spuren mit zeitlicher Relation und symbolischer Relation,
Klasse 5	: Spuren mit zeitlicher Relation und symbolischer Realtion, nicht zeitkonsumierend

Beispiele:

Phonetische Segmentation (Klasse 4)

```
MAU: 0 2879 -1 <p:>
MAU: 2880 639 -1 <nib>
MAU: 3520 479 0 t
MAU: 4000 959 0 S
```

```
MAU: 4960 1919 0 y:
MAU: 6880 2879 0 s
MAU: 9760 639 -1 <nib>
MAU: 10400 1119 -1 <p:>
```

Prosodische Labelung (Grenzen und Akzente; Klasse 5)

```
PRB:    6532    -1      TON: H*; FUN: PA
PRB:    10083   -1      BRE: B3; TON: ?%
```

4.5.3 Wieviele Tiers gibt es im Partitur Format?

Beliebig viele. Das Format wurde so gestaltet, daß es jederzeit erweitert werden kann, wenn eine neue symbolische Information auftaucht (*open format*). Technisch ist dazu nötig, daß man ein neues, noch unbenutztes Label aussucht, eine Syntax und Semantik für die Zeilen entwirft und die Informationen nach diesen Vorgaben formatiert. Dann hängt man mit einem einfach UNIX-Kommando (cat) den neuen Zeilenblock einfach an die bereitsbestehende Partitur Datei an, z.B.

```
cip1 % cat NeuerBlock >> PartiturFile.par
```

Natürlich sollte man nicht vergessen, seine neue Definition eines Tiers an das BAS weiterzugeben, damit sie dort registriert und sorgfältig dokumentiert wird.

4.5.4 Was passiert mit meiner Software, wenn ich einfach das Format erweitere?

Im Normalfall gar nichts. Alle Programme, die auf Partitur Dateien zugreifen, sollten so geschrieben sein, daß die nur die Zeilen herausfiltern (z.B. mit dem UNIX Befehl 'grep'), die sie wirklich brauchen. Wenn dann unten noch irgendwelche dem Programm unbekannten Zeilenblöcke hängen, werden diese einfach ignoriert.

Hier ist ein Beispiel für eine (kurze) komplette Partitur Datei:

```
LHD: Partitur 1.2.3
REP: Muenchen
SNB: 2
SAM: 16000
SBF: 01
SSB: 16
NCH: 1
SPN: KAP
LBD:
TRL: 0 <#Klicken>
TRL: 0 tsch"u"s
TRL: 0 <#Klicken> .
KAN: -1 <nib>
KAN: 0 tS'y:s
KAN: -1 <nib>
DAS: 0 @(BYE BA)
PRB:  6532  -1 TON: H*; FUN: PA
PRB:  10083  -1 BRE: B3; TON: ?%
```

```
WOR:  0  2615  -1 <#>
WOR:  2616  1249  -1 <#Klicken>
WOR:  3866  6217  -1 tsch"u"s
WOR:  10084  1349  -1 <#>
WOR:  11434  291  -1 <P>
MAU: 0 2879 -1 <p:>
MAU: 2880 639 -1 <nib>
MAU: 3520 479 0 t
MAU: 4000 959 0 S
MAU: 4960 1919 0 y:
MAU: 6880 2879 0 s
MAU: 9760 639 -1 <nib>
MAU: 10400 1119 -1 <p:>
TR2: 0 <#Klicken> tsch"u"s . <#Klicken>
ORT: 0 tsch"u"s
```

4.6 Wie arbeitet man effektiv mit Partituren?

Das wichtigste ist natürlich die Beherrschung elementarer Werkzeuge zur *zeilenorientierte Bearbeitung* von Text-Dateien. Daher werden wir uns in den nächsten beiden Doppelstunden intensiv mit solchen Werkzeugen auseinandersetzen.

Dann braucht man immer ein *Visualisierungs-Tool*, d.h. ein Programm, das eine Partitur und das zugehörige Signal liest und geordnet auf einem Display ausgibt, evtl. sogar erlaubt diese dann zu manipulieren.

Das ist an unserem Lehrstuhl das UNIX-Programm par2sfs, da wir zur Darstellung die SFS Software verwenden.