Using linguistic databases for psycholinguistic, phonetic, and phonological research words

Robert Felty
University of Michigan
robfelty@umich.edu

February 1, 2007

Lexical statistics

Lexical Frequency

wordform frequency

only specific instances of a word are counted, e.g. *walked*

lemma frequency

All forms of a word are summed together, e.g. *walk*, *walked* and *walking* are all counted for *walk*

log frequency

Many researchers use log frequency because effects of frequency tend to scale on linearly on a log-linear scale

One of the more common algorithms to calculate log frequency is given in Newman et al. (1997) — first add 1 to any words with frequency of 0 (because log(0) is undefined. Then multiple by 10, then take the base10 log.

Lexical Frequency

- Many experiments have shown that high frequency words are recognized more quickly and accurately in a variety of tasks (lexical decision, cross-modal priming, phonemic restoration, spoken word recognition, word naming, eye-tracking)
- There is still a debate whether this is due to a response bias or an increased sensitivity (see Broadbent, 1967, for an excellent discussion using tube models)
- This is categorized as a facilitatory effect

Neighborhood density

Number of neighbors (Neighborhood density)

the number of words differing by one phoneme (deletions, insertions and substitutions)

Neighborhood frequency

(frequency weighted neighborhood density)

the sum of the frequencies of all edit distance 1 neighbors

Phonetic neighborhood density

(frequency weighted neighborhood probability)

a probability based on confusion matrices (Luce, 1986; Luce and Pisoni, 1998)

$$\sum_{j=1}^{nn} \left\{ \left[\prod_{i=1}^{n} p(PN_{ij} | PS_i) \right] \cdot Freq_{Nj} \right\}$$

Neighborhood density

- Numerous experiments (begun in David Pisoni's lab in the 1980's) have shown that words in dense neighborhoods are recognized more slowly and with less accuracy than words in sparse neighborhoods (including lexical decision, cross-modal priming, spoken word recognition)
- This is categorized as an inhibitory effect

Phonotactic probability

- Phonotactic probability is a measure of the likelihood of certain phonemes appearing in a certain order
- One frequently used measure is from Vitevitch and Luce (2004), in which they calculate a positional probability and a biphone positional probability.
  - The positional probability is simply the sum of the probabilities that a given phoneme will occur in a certain position in a word
  - the biphone positional probability is the sum of transitional probabilities that two phonemes appear in a given position
  - The advantage of this method is that it does not rely on any particular phonological theory
  - The disadvantage is that it does not consider phonology
- Another approach divides syllables into onsets and rimes, and calculates the probability that a given onset and rime occur together Coleman and Pierrehumbert (1997)

Phonotactic probability

- Experiments using phonotactic probability have only shown very small effects, and some of these results are quite disputed

Databases

Overview

There are several publicly available databases

- CELEX (Baayen and Rijn, 1993) (English, German, Dutch)
- Hoosier Mental Lexicon (HML) (Nusbaum et al., 1984) (English)
- MRC Psycholinguistic Database (English) uses Kučera-Francis frequencies
- iPhod (English) uses CMU transcriptions, Kučera-Francis frequencies. Has density, frequency, and phonotactic information

HML

- Based off of the 20,000 word Websters electronic pocket dictionary (English)
- Frequency information taken from Kučera and Francis (1967)
- Word familiarity scores collected primarily from IU students
- Several different density measures are also provided

HML

- Based off of the 20,000 word Websters electronic pocket dictionary (English)
- Frequency information taken from Kučera and Francis (1967)
- Word familiarity scores collected primarily from IU students
- Several different density measures are also provided
- Advantages:
  - freely available
  - Phonetic transcriptions and frequency information based on American English

HML

- Based off of the 20,000 word Websters electronic pocket dictionary (English)
- Frequency information taken from Kučera and Francis (1967)
- Word familiarity scores collected primarily from IU students
- Several different density measures are also provided
- Advantages:
  - freely available
  - Phonetic transcriptions and frequency information based on American English
- Disadvantages:
  - based off of relatively small (1 million words) and old (40 years) corpus
  - missing many common words
  - only lists lemma entries

Structure of HML

- HML is contained all in one fixed-width ASCII text file, with one entry per line
- Has its own relatively intuitive phonetic transcription system using a one character per phoneme mapping
- contains the following fields:

Structure of HML

1. Orthography
2. Transcription
3. Syntactic Code
4. Content Function
5. CV Pattern
6. Length
7. Frequency (K & F)
8. Log Frequency (K & F)
9. Familiarity
10. Density A
11. Mean Frequency A
12. Standard Dev. Mean Freq. A
13. Mean Log Freq. A
14. Standard Dev. Mean Log Freq. A
15. Density B
16. Mean Frequency B
17. Standard Dev. Mean Freq. B
18. Mean Log Freq. B
19. Standard Dev. Mean Log Freq. B

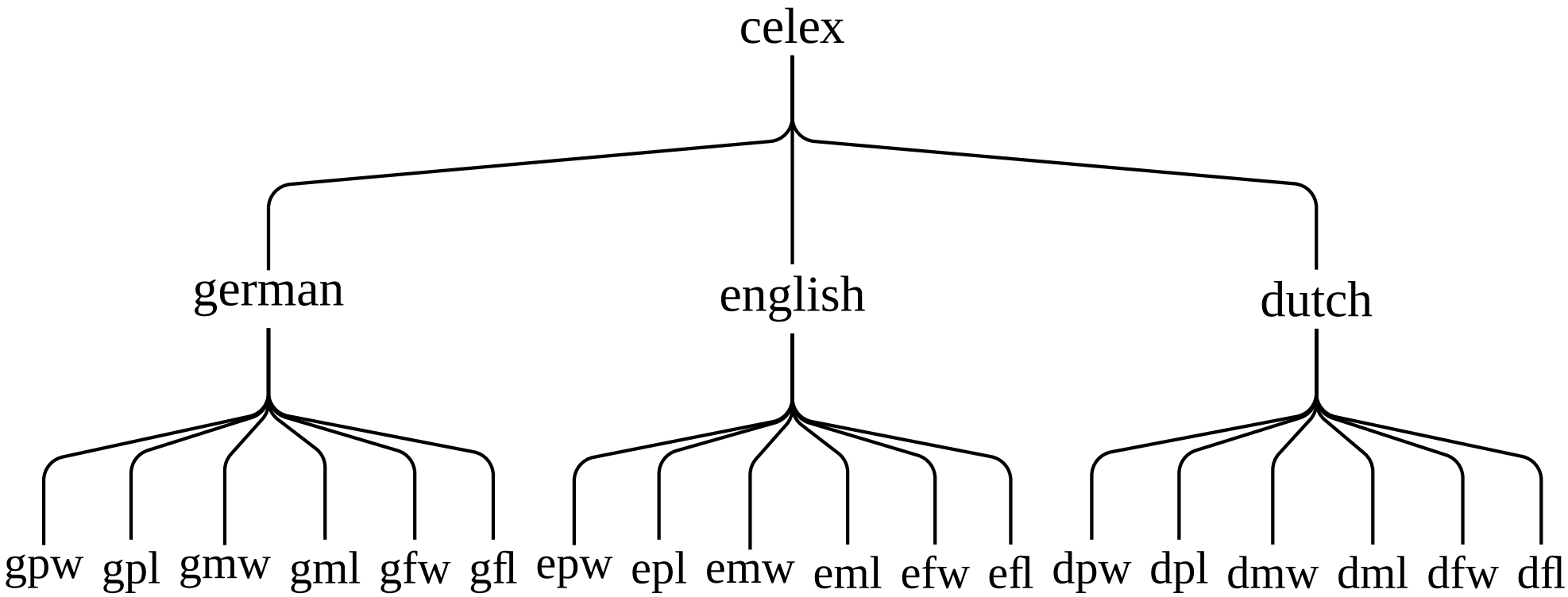
CELEX

- CELEX contains a variety of phonetic, morphological, and frequency information for German, Dutch, and English (British)
- Frequency information based on separate corpora for each language
  - English — 17.9 million words
  - German — 6.1 million words
  - Dutch — 42 million words

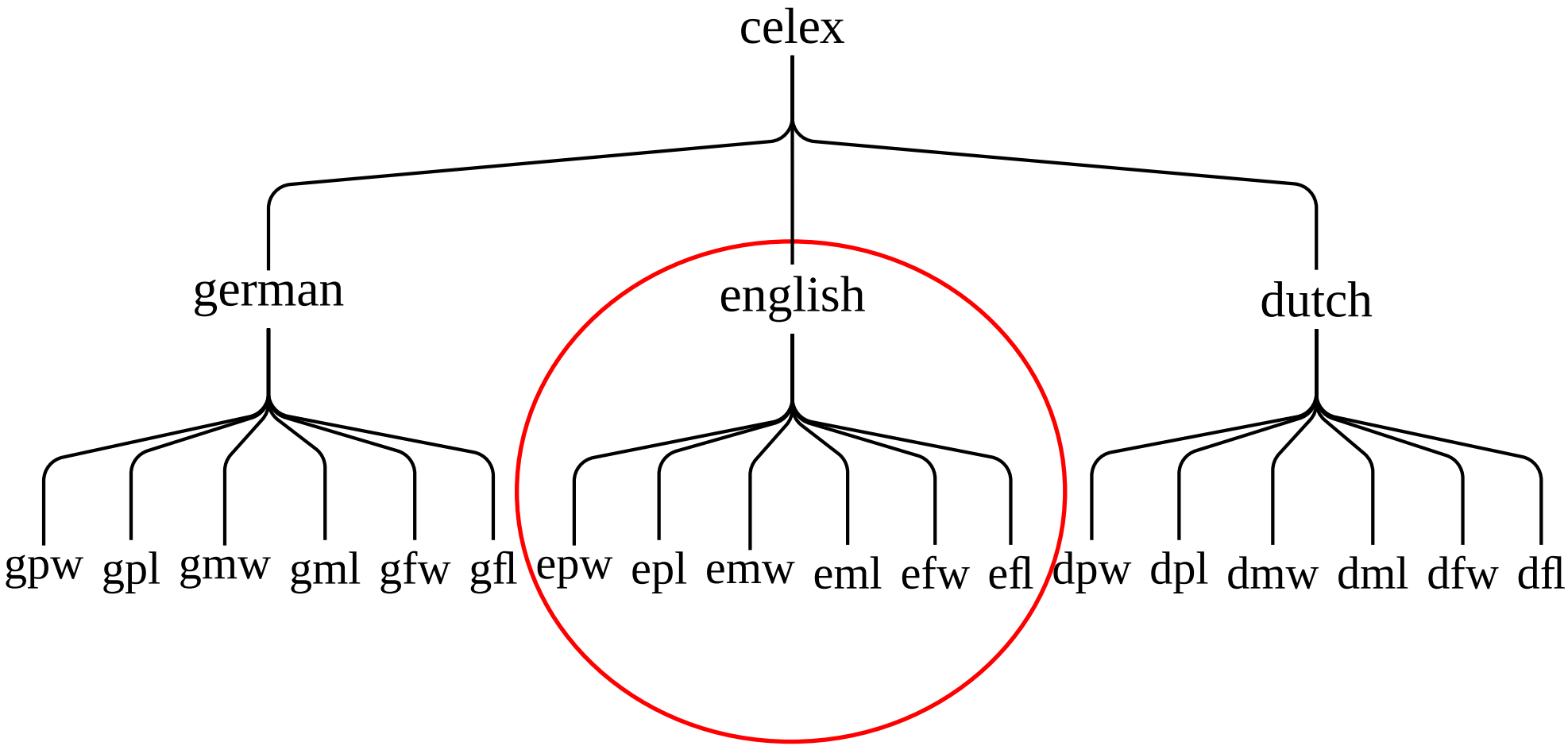
CELEX

- Advantages:
  - contains both wordform and lemma frequencies
  - contains additional morphological information
  - contains a number of different frequency measures, including broken down by written and spoken sources
  - Based off of large and fairly recent corpora
- Disadvantages:
  - does not contain word familiarity ratings
  - English portion is based off of British English
  - somewhat complicated structure — not very user friendly

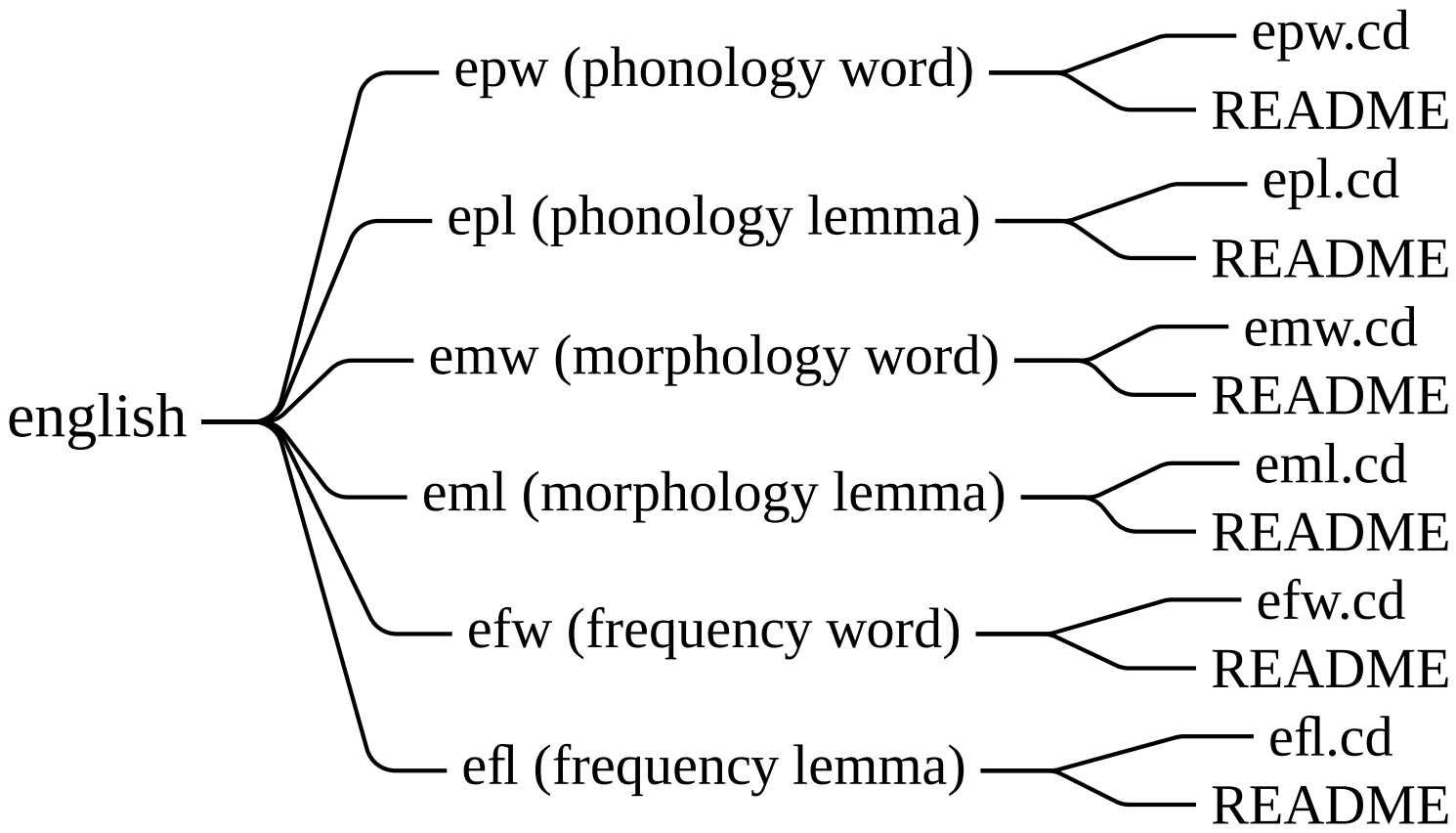
Structure of CELEX



Structure of CELEX



Structure of CELEX



Structure of CELEX (continued)

- There is a rather extensive manual in the main directory, named intro_let.ps (let = letter paper, there is also intro_a4.ps, for a4 size paper)
- You might want to convert the .ps (postscript) file into pdf using the command:

```
ps2pdf intro_let.ps
```

Structure of CELEX (continued)

ENGLISH PHONOLOGY, WORDFORMS

The epw.cd file contains the following fields:

1. IdNum
2. Word
3. Cob
4. IdNumLemma
5. PronCnt
6. PronStatus
7. PhonStrsDISC
8. PhonCVBr
9. PhonSylBCLX

Structure of CELEX (continued)

Those words which appear with alternative pronunciations are assigned 4 extra fields for each pronuncation. For instance, the columns

10. PronStatus
11. PhonStrsDISC
12. PhonCVBr
13. PhonSylBCLX

specify the second pronunciation variant, if present. the third variant, if present, occupies columns 14-17, etc.

Structure of CELEX (continued)

For 48 words, the number of different pronunciations was too large (greater than 23) to allow listing of all variants on a single line without exceeding the AWK limit of 100 fields per line. These words appear in epw.cd with their first 23 variants. Additional variants — the maximum number of variants is 60, for ”proportional-representation” — are available in the directory ”variants”. This directory contains the files

epw.cd Excerpt

```
1922\allot\2\1092\1\P\@-'lQt\[V][CVC]\[@][lOt]
1923\allotment\20\1093\1\P\@-'lQt-m@nt\[V][CVC][CVCC]\[@][lOt][m@nt]
1924\allotments\17\1093\1\P\@-'lQt-m@nts\[V][CVC][CVCCC]\[@][lOt][m@nts]
1925\allots\1\1092\1\P\@-'lQts\[V][CVCC]\[@][lOts]
1926\allotted\17\1092\1\P\@-'lQ-tId\[V][CV][CVC]\[@][lO][tId]
1927\allotting\3\1092\1\P\@-'lQ-tIN\[V][CV][CVC]\[@][lO][tIN]
1928\all-out\51\1095\1\P\'$l-6t\[VVC][VVC]\[O:l][aUt]
1929\all out\0\1094\1\P\"$l-'6t\[VVC][VVC]\[O:l][aUt]
1930\all over\0\1096\1\P\"$l-'5-v@R\[VVC][VV][CVC]\[O:l][@U][v@r*]
1931\allow\393\1097\1\P\@-'l6\[V][CVV]\[@][laU]
1932\allowable\20\1098\1\P\@-'l6-@-bP\[V][CVV][V][CS]\[@][laU][@][bl,]
1933\allowably\0\1099\1\P\@-'l6-@-blI\[V][CVV][V][CCV]\[@][laU][@][blI]
1934\allowance\321\1100\1\P\@-'l6-@ns\[V][CVV][VCC]\[@][laU][@ns]
1935\allowances\209\1100\1\P\@-'l6-@n-sIz\[V][CVV][VC][CVC]\[@][laU][@n][sIz
1936\allowed\474\1097\1\P\@-'l6d\[V][CVVC]\[@][laUd]
```

Tools

Publicly available tools

- MRC Psycholinguistic Database
- Collection of python scripts for using the HML
  `http://aix1.uottawa.ca/~jmielke/topics/`
- CELEX online
- Washington Univ. neighborhood database - uses HML
- LINGUA — corpus independent tool for calculating frequency and *orthographic* neighborhood, as well as generating nonwords (download)
- Online phonotactic probability calculator — computes positional probability and biphone probability using the HML
- iPhod online and downloadable version, uses CMU transcription and Kucera-Francis Frequencies. Contains frequency, density, and phonotactic information Vitevitch and Luce (2004)

Home brewed tools

- Useful programs: grep, perl, awk, sed, Matlab
- all can use regular expressions extensively
- not very helpful: Excel — database files are simply too big

Where can I get these tools?

- grep, perl, awk, and sed are standard programs on any UNIX-based operating system (UNIX, Linux, Mac OSX – i.e. most everything but Windows)
- These are available for Windows, but are a bit of a pain to install. If you are interested in installing them on your own machine, investigate Cygwin
- Otherwise, I recommend using a publicly available Mac, or logging in remotely to the University servers
- Matlab is available on most public computers on Campus (Windows, Mac, and Linux). Unlike the other programs, it is not free.

Regular Expressions

- Finding the information you need from the databases will require the use of regular expressions
- Regular expressions are a feature in many programming languages that allow one to search for a given string in a body of text, including the use of some special characters
- Problem: I want to find all CVC words in the English CELEX database

Regular Expressions

- Finding the information you need from the databases will require the use of regular expressions
- Regular expressions are a feature in many programming languages that allow one to search for a given string in a body of text, including the use of some special characters
- Problem: I want to find all CVC words in the English CELEX database
  Solution: `grep -E '\\\[CVC\]\\' epw.cd`

Regular Expressions

- Finding the information you need from the databases will require the use of regular expressions
- Regular expressions are a feature in many programming languages that allow one to search for a given string in a body of text, including the use of some special characters
- Problem: I want to find all CVC words in the English CELEX database
  Solution: `grep -E '\\\[CVC\]\\' epw.cd`
- Problem: I want to know how many words that start and end with the letter *k*

Regular Expressions

- Finding the information you need from the databases will require the use of regular expressions
- Regular expressions are a feature in many programming languages that allow one to search for a given string in a body of text, including the use of some special characters
- Problem: I want to find all CVC words in the English CELEX database
  Solution: `grep -E '\\\[CVC\]\\' epw.cd`
- Problem: I want to know how many words that start and end with the letter *k*
  Solution: `grep -iEc '\\k[a-z]*k\\' epw.cd`

Regular Expressions

Special characters:

`. ? + * [] {} () | ^ $ \`

character classes and anything

- `.` matches any character
- `[]` matches any of the characters within the brackets e.g. `[a0]` matches both *a* and *0*
- several predefined shortcuts are also possible
  - `[a-z]` matches all lowercase letters
  - `[A-Z]` matches all uppercase letters
  - `[a-zA-Z]` matches all uppercase and lowercase letters
  - `[0-9]` matches all numbers

Regular Expressions

Special characters:

```
. ? + * [] {} () | ^ $ \
```

Quantifiers

- `?` matches 1 or 0 of the preceding character, e.g. `colou?r` matches *color* and *colour*
- `+` matches 1 or more of the preceding character, e.g. `bug +off` matches *bug off*, *bug off*, but not *bugoff*
- `*` matches any number of the preceding character, e.g. `colou*r` matches *color*, *colour*, *colouur* and so on
- `{}` used to specify the number of times a character should be matched. Ranges are also possible. Examples:
  - `a{2}` matches only *aa*
  - `[a-z]{2}` matches two lowercase letters, e.g. *ab*
  - `[a-z]{2,4}` matches 2–4 lowercase letters, e.g. *al* or *foo*

Regular Expressions

Special characters:

```
. ? + * [] {} () | ^ $ \
```

Grouping

- `()` used to group sequences. Useful especially for backreferences (more on that later), and
- `|` used as an or operator, e.g. `x|y` matches either *x* or *y*
- Parentheses and `|` can be used together for very nice effects. For example, say I want to find common typos involving duplicate words (such as *a a* or *the the*). I could write an expression like so `(a|the) \1`
  which says “match either *a* or *the* followed by a space followed by whatever was matched in the parentheses”
- `\1` is a backreference. You can use multiple backreferences of the form `\n` where n is the nth pair of parentheses in the expression.

Regular Expressions

Special characters:

`. ? + * [] {} () | ^ $ \`

The beginning, the end, and escaping

- `^` matches the beginning of the string
  Within brackets, negates the pattern, e.g. `[^xy]` matches everything but *x* or *y*
- `$` matches the end of the string
- `\` is the escape character. When you want to use one of the special characters as a normal character, it must be preceded by `\`

Grep specific information

- grep will search a file on a line by line basis, and return any lines which contain the regular expression
- In the case of CELEX, we will take advantage of the fact that fields are separated by\

Grep specific information

Like many UNIX programs, grep has quite a few options available. For a complete list, type `man grep`

- -E extended regular expressions — allows us to use all the special characters
- -i ignore case
- -c simply print the number of matches
- -v invert match, i.e. return everything that does not match the expression

These can be used in conjunction with one another, e.g.

```
grep -icv 'dog' file
```

returns the number of lines that do not contain the word dog from the file 'file'.

Regular Expression practice

Practice writing some regular expressions that will find the following from CELEX:

- all words begin with ‘st’
- all words that end in ‘ing’
- word that begin with ‘st’ and ending with ‘ing’
- all monosyllabic words
- all disyllabic words
- Find all words with a frequency of greater than 23

Regular Expression practice

Practice writing some regular expressions that will find the following from CELEX:

- all words begin with ‘st’
  `\\st`
- all words that end in ‘ing’
- word that begin with ‘st’ and ending with ‘ing’
- all monosyllabic words
- all disyllabic words
- Find all words with a frequency of greater than 23

Regular Expression practice

Practice writing some regular expressions that will find the following from CELEX:

- all words begin with ‘st’
  `\\st`
- all words that end in ‘ing’
  `ing\\`
- word that begin with ‘st’ and ending with ‘ing’
- all monosyllabic words
- all disyllabic words
- Find all words with a frequency of greater than 23

Regular Expression practice

Practice writing some regular expressions that will find the following from CELEX:

- all words begin with ‘st’
  `\\st`
- all words that end in ‘ing’
  `ing\\`
- word that begin with ‘st’ and ending with ‘ing’
  `\\st[a-z]*ing\\`
- all monosyllabic words
- all disyllabic words
- Find all words with a frequency of greater than 23

Regular Expression practice

Practice writing some regular expressions that will find the following from CELEX:

- all words begin with ‘st’
  `\\st`
- all words that end in ‘ing’
  `ing\\`
- word that begin with ‘st’ and ending with ‘ing’
  `\\st[a-z]*ing\\`
- all monosyllabic words
  `\\\[[CV]+\]\\`
- all disyllabic words

- Find all words with a frequency of greater than 23

Regular Expression practice

Practice writing some regular expressions that will find the following from CELEX:

- all words begin with ‘st’
  `\\st`
- all words that end in ‘ing’
  `ing\\`
- word that begin with ‘st’ and ending with ‘ing’
  `\\st[a-z]*ing\\`
- all monosyllabic words
  `\\\[[CV]+\]\\`
- all disyllabic words
  `\\\[[CV]+\]\[[CV]+\]\\`
- Find all words with a frequency of greater than 23

Regular Expression practice

Practice writing some regular expressions that will find the following from CELEX:

- all words begin with ‘st’
  `\\st`
- all words that end in ‘ing’
  `ing\\`
- word that begin with ‘st’ and ending with ‘ing’
  `\\st[a-z]*ing\\`
- all monosyllabic words
  `\\\[[CV]+\]\\`
- all disyllabic words
  `\\\[[CV]+\]\[[CV]+\]\\`
- Find all words with a frequency of greater than 23
  ???

Perl specific information

Whereas grep is designed to perform one specific task, perl is much more powerful, but also more complicated.

- perl is a scripted language, meaning that it does not need to be compiled such as C or JAVA
- perl is very good at handling text and file input and output, which is primarily what we will be doing
- perl can also very easily access any other standard commands through the use of backticks, e.g. my `$return = ‘grep rob epw.cd‘`
- perl has a fairly simple syntax, which draws mostly from C, awk, sed and shell scripting
- perl’s motto is “there’s more than one way to do it”

Perl specific information

There is a wealth of perl documentation on the web. Here is one handy reference on perl regular expressions: http://perldoc.perl.org/perlreref.html or type

```
perldoc perlreref
```

for the same info in your terminal (try also

```
perldoc perlre
perldoc perlop
```

Perl variable types

Perl has three main variable types:

- scalars — a single thing (can be a number, string, boolean etc.) indicated by a $, e.g.

```
$perl = 'a powerful and easy programming
language';
```

- arrays — a collection of things indexed by integer starting at 0 indicated by a @, e.g.

```
@perls = (3,'blind','mice')
print @perls;
    3blindmice
```

- hashes — a collection of things indexed by custom, unique keys indicated by %, e.g.

```
%favorites = (color => 'green',food => 'thai tofu curry');
while ( my ($key, $value) = each(%favorites) ) {
  print "$key => $value\n";
}
```

Commonly used functions

- matching `m/regex/flags;`
- substitution `s/regex/replacement/flags;`
- splitting a file into columns `split /regex/flags;`

Frequency Example

Let’s return to our problem of finding words with frequency of less than 23

```
 1   #!/usr/bin/perl -w
 2   #this script takes one command line argument -
 3   #id1 is the file we want to look at
 4   #it returns all words with frequency of less than 23
 5
 6   # the name of the file will be passed as the first command line argument
 7   my $id1 = $ARGV[0];
 8
 9   #initialize some variables
10   my %entry;
11   my $index1=0;
12   open(ID1, $id1) || die("could not open"); #open the file
```

Frequency Example

```
13   #read the file line by line and store contents into an array of hashes
14   while (<ID1>) {
15     chomp; #strip off newline characters
16     my ($id, $spell, $freq, $lemmaId, $pronCnt, $pronStat, $phon, $CVC, $phon2) = split /\\/;
17     $entry[$index1]{'id'}=$id;
18     $entry[$index1]{'spell'}=$spell;
19     $entry[$index1]{'freq'}=$freq;
20     $entry[$index1]{'lemmaId'}=$lemmaId;
21     $entry[$index1]{'pronCnt'}=$pronCnt;
22     $entry[$index1]{'pronStat'}=$pronStat;
23     $entry[$index1]{'phon'}=$phon;
24     $entry[$index1]{'CVC'}=$CVC;
25     $entry[$index1]{'phon2'}=$phon2;
26     $index1++;
27   }
28   close(ID1);
29   for (my $i=0;$i<$index1;$i++) {
30     if ($entry[$i]{'freq'}<23) {
31       for my $key ( keys %{ $entry[$i] } ) {
32         print "$entry[$i]{$key}\\";
33       }
34     print "\n";
35     }
36   }
```

Rob's perl scripts

- I am giving you access the CELEX database and many of the perl scripts which I have written to interact with CELEX at `/afs/umich.edu/user/r/o/robfelty/Public/celex`
- The perl files are in `celex/pl`
- I have tried to name them appropriately according to what they do and comment them fairly well, but I provide no guarantees
- I am also providing six files with recomputed CELEX values — two for each language. (in `celex/recomputed`)
  - [ged]pwRecomputed.cd has summed frequencies for words with multiple entries in CELEX, e.g. *walking* it listed both as a noun and a verb. The recomputed values sums these and contains only one entry. This is probably more appropriate for both auditory and visual lexical access research
  - [ged]pwSummedHomophones.cd has in addition summed the frequencies for homophones. This is more appropriate for auditory research.

Matlab specific information

- Unlike perl and grep, Matlab is an interactive programming environment. What this means is that instead of processing files in batches, you can load data into memory and interact it with directly.
- Matlab also allows you to write scripts and functions much like perl does
- Matlab also has many built-in tools for doing statistics and making plots
- Matlab also has a very nice debugger, which allows you to stop a script at specified points so that you can see exactly what is being done
- Finally, Matlab has very handy built-in help files

José and Rob's Matlab scripts

- José Benkí and I have put together numerous scripts to interact with the HML, to run speech-in-noise experiments, and to analyze the data from such experiments
- I have put the HML files in `/afs/umich.edu/user/r/o/robfelty/Public/hml` and the matlab files in `hml/matlab`
- The main one we will start off is ReadDict, which loads the entire HML database into memory
- Once that is loaded, we'll go over how to do some of our previous examples in Matlab

References

Baayen, R., H and H. Rijn. 1993. The CELEX lexical database (cd-rom).

Broadbent, D. 1967. Word-frequency effect and response bias, *Psychological Review*, 74, 1–15.

Coleman, John and Janet Pierrehumbert. 1997. Stochastic phonological grammars and acceptability, in John Coleman, (Ed.) *Computational phonology: Third meeting of the ACL special interest group in computational phonology*, Association for Computational Linguistics, Somerset, NJ: Association for Computational Linguistics, 49–56.

Kučera, F. and W. Francis. 1967. *Computational Analysis of Present Day American English*, Providence: Brown University Press.

Luce, Paul. 1986. *Neighborhoods of words in the mental lexicon*, Ph.D. thesis, Indiana University.

Luce, Paul and David Pisoni. 1998. Recognizing spoken words: The neighborhood activation model, *Ear and Hearing*, 19, 1–36.

Newman, Rochelle S., James R. Sawusch, and Paul A. Luce. 1997. Lexical neighborhood effects in phonetic processing, *Journal of Experimental Psychology*, 23(1), 873– 889.

Nusbaum, H. C., D. B. Pisoni, and C. K. Davis. 1984. Sizing up the hoosier mental lexicon: Measuring the familiarity of 20 000 words, Research on Speech Perception Progress Report No. 10 10, Speech Research Laboratory, Psychology Department, Indiana University, Bloomington.

Vitevitch, Michael S. and Paul A. Luce. 2004. A web-based interface to calculate phonotactic probability for words and nonwords in english, *Behavior Research Methods, Instruments, & Computers*, 36(3), 481–487.