Ravnur Tools for BLARK creation

Documentation

ver. 0.96

Table of Contents

1. Introduction

This manual covers the Ravnur toolbox. The tools are mainly relevant for projects establishing new BLARKs (or revising existing ones thoroughly). Each tool is presented in a separate section below.

Before consulting this manual, make sure that the Ravnur toolbox is accessible to you. The tools are designed for interactive use with a standard browser (such as Google Chrome). Please contact your technical support concerning the installation of the programs (cf. Appendix) on an internet server near you.

In this document (including appendices) the proper names Ravn and Ravnur are used interchangeably.

1.1 Tools presented in this manual

`ScrambleText`	[makes random scramblings of text lines for reading sessions]
`MakePhoneClusters`	[analyses phonetic forms appearing in a transcription]
`MakeWdList`	[creates phonetically complete word lists for reading sessions]
`MakeLemma`	[suggests new lexical entries (lemmas) based on existing ones]
`PushPrompt`	[supports interactive reading sessions by prompting reader]
`EvalBlark`	[controls formal wellformedness for a completed BLARK]

1.2 Technical notes

All Ravnur tools are stand-alone cgi scripts written in perl (5.0 or later). They will run on any Linux-based internet server (or similar platforms) without modifications; you may however need to adjust program-internal file paths and globs as required by your local file system. Output is pure html with no embedded css or js. The tools do not require (or support) user identification.

In all scripts, user-specific settings are located in section ”`settings`” and may be adjusted to taste. The color coding of the html output (e.g. `<body bgcolor=#eeeebb>`) was used for version control during the development period and should probably be removed (or better: replaced by ccs-scripting).

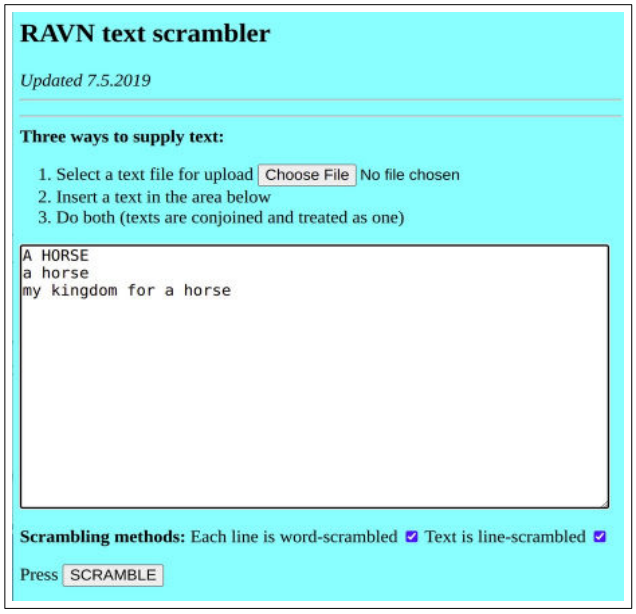
The perl code is found in the Appendix (9.1-9.6).

2. ScrambleText

This tool is used when preparing reading materials for recording sessions.

ScrambleText makes random scramblings of sentences to be used as reading materials, in order to avoid effects of priming and monotony. Its use is demonstrated by an example.

This screen dump is taken right after the user has entered three text lines in the text box (notice that the tickboxes below the text area allow some control of the scrambling procedure). On pressing SCRAMBLE, the program returns a scrambled version of the text input.



Examples:

```
kingdom my for a horse
horse a
A HORSE
```

```
HORSE A
horse a my for kingdom
horse a
```

Input: Dictionary *or* a text of words occurring in Dictionary
Output: The input text scrambled (randomly or otherwise)

---

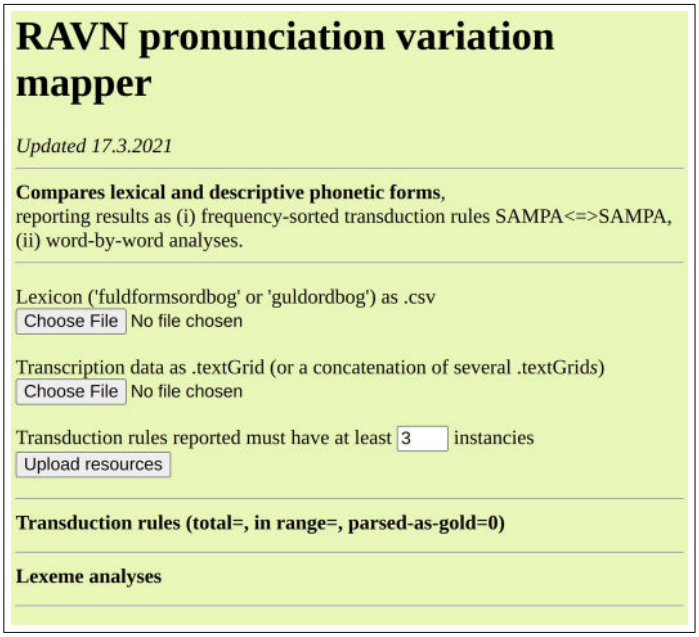
3. MakePhoneClusters

This tool is used when developing a BLARK transcription corpus.

The tool compares reading materials (a text) to the corresponding phonetic transcription (a particular reading of the text). The tool can thus assist the transcriber working on phonetic transcriptions of speech recordings by comparing the sound-based transcribed form to the lexical form (as represented in the Dictionary). More specifically, the program outputs all discrepancies between the corresponding lexical phonetic forms (LPF) and descriptive phonetic forms (DPF).

As explained elsewhere, a formally wellformed Dictionary (status ”ABLARK”) comprises all lexical forms appearing in the text materials included in the BLARK (with the exception of the Background Text Corpus).

This screen shot shows the opening page. From here the user enters a (ABLARK-compliant) Dictionary and a transcription file (in TextGrid-format).



The script returns two output files: (i) a LPF-to-DPF map showing the the transduction rules from LPF-forms to DPF-forms, (ii) a word-by-word analysis showing, for each lexical word, its lexical-phonetic forms and descriptive-phonetic forms aligned.

The output is csv-formatted for easy transfer to e.g. data tables or statistical analysis.

Input: TextGrid (transcription); Dictionary

Output: Pronunciations deviating from lexical form

---

4. MakeWdList

This tool is used when preparing phonetically controlled reading materials.

The MakeWdList script is used for creating (random) lists of words, phonetically complete in the sense that the words collectively cover all the SAMPA-phones (based on their lexicalized pronunciations). Such word lists are convenient as reading materials for speech technology (speech recordings for TTS and ASR training data).

This screen dump shows the opening page. From here the user uploads a text file (typically containing several thousands text words) and a Dictionary (covering all tokens in the text file).

On pressing UPLOAD LEXICON, the user receives a list of words picked randomly among the text words, such that all phones in the phonetic domain (cf. ACCEPTED PHONE SYMBOLS) are represented in the lexical-phonetic rendering of the wordlist.

The output format can be specified using the “Word length” feature.

NOTE: The strange Arabic-looking symbols in the screen shot (cf. ACCEPTED GRAPHEMES) are due to a flawed browser-setting unrelated to the server program. Please ignore.

Input: Dictionary; SAMPA definition table.

Output: A phonetically complete wordlist.

---

5. MakeLemma

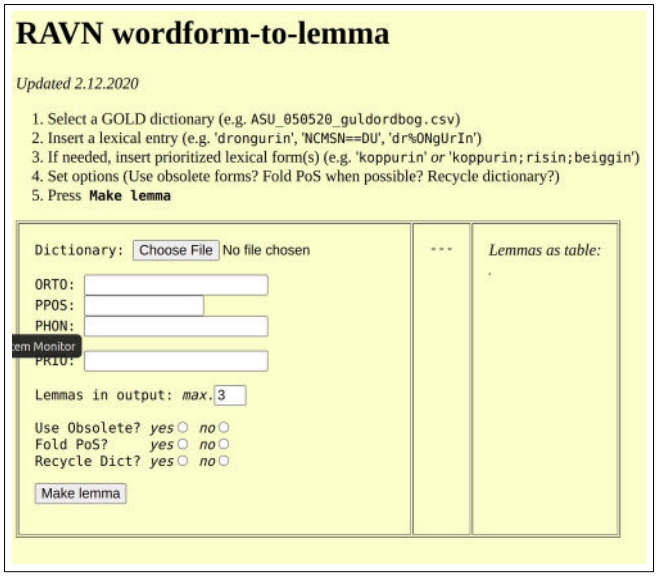
This tool is used when creating or expanding a Dictionary.

The BLARK Dictionary is a full-form lexicon. Each lemma is thus represented by all its inflected forms, as shown in these examples.

```
---
ORTO:annarhvør       PPOS:PI-MSN--U        PHON:%an:arkv~2:r
ORTO:annanhvønn      PPOS:PI-MSA--U        PHON:%an:ankv~9n:
ORTO:øðrumhvørjum    PPOS:PI-[MN]SD--U     PHON:%2:rUnkv~9rjUn
ORTO:onnurhvør       PPOS:PI-FSN--U        PHON:%On:Urkv~2:r
ORTO:aðruhvørja      PPOS:PI-FSA--U        PHON:%EA:rUkv~9rja
ORTO:aðruhvørjari    PPOS:PI-FSD--U        PHON:%EA:rUkv~9rjarI
ORTO:annaðhvørt      PPOS:PI-NS[AN]--U     PHON:%an:arkv~9zd
ORTO:aðrirhvørjir    PPOS:PI-MPN--U        PHON:%EA:rIrkv~2:rjIr
ORTO:aðrirhvørjar    PPOS:PI-MPA--U        PHON:%EA:rIrkv~2:rjar
ORTO:øðrumhvørjum    PPOS:PI-[FMN]PD--U    PHON:%2:rUnkv~9rjUn
ORTO:aðrarhvørjar    PPOS:PI-FP[AN]--U     PHON:%EA:rarkv~2:rjar
ORTO:onnurhvørji     PPOS:PI-NP[AN]--U     PHON:%On:Urkv~2:rjI
---
ORTO:báðir           PPOS:PI-MPN--U        PHON:b%OA:jIr
ORTO:báðar           PPOS:PI-MPA--U        PHON:b%o:ar     PHON:b%u:war
ORTO:báðum           PPOS:PI-[FMN]PD--U    PHON:b%OA:vUn
ORTO:beggja          PPOS:PI-[FMN]PG--O    PHON:b%Ed:Za
ORTO:báðar           PPOS:PI-FP[AN]--U     PHON:b%o:ar     PHON:b%u:war
ORTO:bæði            PPOS:PI-NP[AN]--U     PHON:b%EA:jI    PHON:b%OA:jI
---
```

Lemmas are separated by lines containing only '`---`'. Blank lines are allowed everywhere. A word form entry (i.e. one line in the Dictionary file) contains the orthographic form (`ORTO`), part-of-speech (`PPOS`) and at least one phonetic rendering (`PHON`).

The user must specify (*i*) a Dictionary; (*ii*) a word form WF, given by either its orthographic form, phonetic form, PoS, or any combinations thereof (typically, the user will want to specify at least the orthographic form); (*iii*) optionally, one or more prioritized word forms PF1..PF*n*; (*iv*) various setttings controlling the search. Based on the user input, the program searches for lexical forms F1..SF*m* in Dictionary that are morphologically analogous to WF. Based on the various forms, the program presents a number of construed lemmas (CL1..CL*k*) as suggestions for a new lexical entry for WF. In case one or more PFs are specified, suggestions containing PF1..PF*n* are given absolute priority. By way on an example, for a WF '', PF could be ''.



The tool MakeLemma is used for creating new lexical entries. As the BLARK Dictionary is a full-form lexicon (each lemma represented by all its inflected forms), new entries can be cumbersome to type in. MakeLemma expands a single wordform (inserted by the user along with PoS-valuesom and phonetic form) into a fully-fledged lemma derived from the existing Dictionary by analogous

reasoning. MakeLemma typically produces several alternative suggestions for lemmas (sorted by likelihood).

Input: A word form (representing a lemma *L*); Dictionary.

Output: Fully developed lemmas (suggestions for *L*)

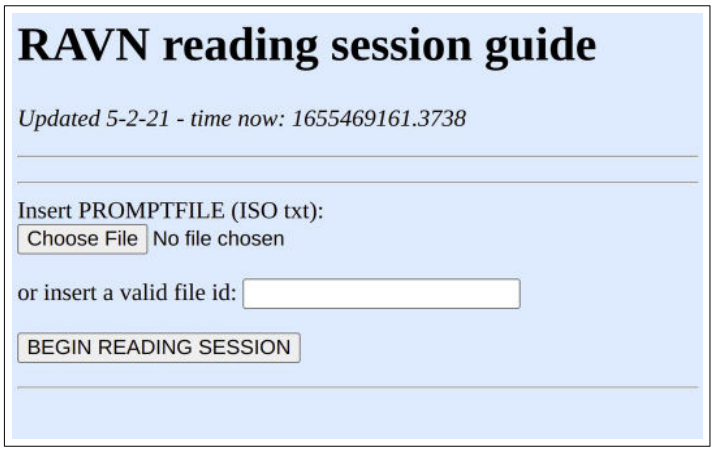
---

6. PushPrompt

This tool is used during a recording session with an informant reading aloud.

PushPromt presents the text items (words, sentences etc.) in the reading material to the reader, allowing her to manage the recording session interactively (adjusting her reading tempo, repeating speech productions at wish, inserting short breaks as needed, etc.). When the reading session is completed, a log file (with time stamps for each production) is presented as a data table.

A PushPrompt session thus falls in three stages: (*i*) the operator prepares the session by uploading the reading text; (*ii*) the informant reads each text sample aloud, and presses the NEXT key when satisfied (bad readings can be corrected on-the-fly); (*iii*) the operator concludes the reading session by downloading the log file with time codes (as Epoch timestamps). The start page (stage 1) is seen to the right.



The three-stage procedure described supports a practical setup with just a single operator supervising the informant. The recording device (preferably using a directional mic in low-echoic surroundings, or even a sound studio) can be left unattended during the whole session, allowing the operator to concentrate on the performance of the informant. The informant controls the text shifts, and may speed up or slow down at her own convenience. All recorded material will of course be contained in a single wav-file, typically a long one with many irrelevant passages; but isolating the relevant passages can be done automatically (without human supervision) based on the timestamps.

Input: Reading manuscript; Dictionary.

Output: Data table (time codes etc.); session log.

---

7. EvalBlark

This tool is used for quality assessment of a completed BLARK.

We use the term ”A-BLARK” for a BLARK which is formally consistent by proof. The tool EvalBlark helps the developer establish A-BLARK-hood. Only approved A-BLARKs should be shared publicly.

A wellformed A-BLARK must meet these criteria:

- All letter symbols occurring in the text based resources must be defined in the Global Alphabet
- All phone symbols used in the transcription corpus, the Dictionary, and elsewhere must be defined in the Global Phone Table
- All orthographic (word) forms occurring in the transcription corpus must be represented in the Dictionary
- All PoS-tags appearing in the text based resources (mainly the Dictionary, but possibly elsewhere as well) must be defined in the Global PoS Table

When establishing a new BLARK, the resource files Global Alphabet, Global Phone Table and Global PoS Table should be created prior to all other developments.

EvalBlark accepts as input a diverse collection of BLARK resources (corpora, definition tables, dictionaries, texts, transcriptions, and more), returning a list of formal inconsistences annotated for location, kind and degree of inconsistency, and level of significance. Where possible, EvalBlark also suggests corrections (as an option).

Input: any collection of BLARK language resources

Output: a list of formal inconsistencies

---

8. Concluding remarks

The Ravnur tools are not language specific and can be used without modifications for BLARK development in other languages, as long as the necessary formal definitions are provided (defining tables of alphabetic letters, phonetic symbols and PoS-labels, etc.). It is our hope that the toolbox may help other 'small' languages establish their own BLARK.

A caveat is in place, though: Some of the tools may be less relevant for polysynthetic languages. An obvious example is the tool for creation of new lexical entries, MakeLemma, which was developed with a Germanic language in mind.

9. Appendix

In this section the code for all Ravnur tools is presented.

The scripts contain comment lines explaining the most essential features (especially concerning the user settings).

The scripts can be copied directly into a text file. Simply cut'n'paste the code lines between horizontal lines into a text editor, and rename the file to your taste (perl scripts are traditionally named *`something`*`.pl`).

For further details, please contact the Ravnur group (https://www.maltokni.fo).

9.1 ScrambleText

```
#!/usr/bin/perl
# Program ScrambleText_1_0.cgi
# Distribution license CC-BY (RAVNUR project group, https://www.ravn.fo)

# -------------- INPUT ---------------

read(STDIN,$datax,%ENV{'CONTENT_LENGTH'}); ($bound) = $datax=~/^(--+\S+)/;

for (split /$bound\r?\n?/,$datax) {
  /^on\b/mi and /RAVN([WL])SCRAM/ and $method{$1}=1 and next;
  ($t=$_) =~ s!^content-\w.+\r?\n!!gim;
  $txt.=$t if $t=~/[^\s-]/ and $t=~/\S/;
};

$txt =~ s!\n\n+!\n!g;
$txt =~ s!(.*)!&scrwds($1)!meg if $method{'W'};
$txt = &scrlin($txt) if $method{'L'};

sub scrwds { join " ",&scram( $_[0]=~/(\S+)/g ) };
sub scrlin { join "\n",&scram( $_[0]=~/^(.+)/gm ) };
sub scram { my(@a) = sort map(rand."\t$_", @_); map(s/^\S+\t// && $_, @a) };

# -------------- OUTPUT ----------------

print <<RAVN
Content-type: text/html

<html>
<head></head>
<body bgcolor='#99FFFF'>
<h2>RAVN text scrambler</h2>
<i>Updated 7.5.2019</i>
<hr><pre>$txt</pre><hr>
<form action="https://yourserver.xyz/cgi-bin/Ravn/ScrambleText_1_0.cgi"
      method=POST enctype="multipart/form-data">
  <b>Three ways to supply text:</b><br>
  <ol>
    <li>Select a text file for upload
      <input type="file" name="RAVNLIST" accept=".txt">
    </li>
      <li>Insert a text in the area below</li>
    <li>Do both (texts are conjoined and treated as one)</li>
  </ol>
  <textarea name="RAVNTXT" rows=15 cols=60></textarea><p>
  <b>Scrambling methods:</b>
  Each line is word-scrambled <input type="checkbox" name="RAVNWSCRAM" checked>
  Text is line-scrambled <input type="checkbox" name="RAVNLSCRAM" checked>
  <p>
  Press <input value="SCRAMBLE" type=submit>
</form>
</body>
</html>
RAVN
```

9.2 MakePhoneClusters

```
#!/usr/bin/perl
# Program MakePhoneClusters_1_0.cgi
# Distribution license CC-BY (RAVNUR project group, https://www.ravn.fo)

# ----------------- settings -------------

$lim = 3;   # DEFAULT value: only var-rules with c>=$lim are reported

# -------------- INPUT ----------------

read(STDIN,$datax,%ENV{'CONTENT_LENGTH'});

($bound) = $datax=~/^(--+\S+)/;

for (split /$bound\r?\n?/,$datax) {
  $t=$_;  $t=~s/^\r?\n//; $t=~s/^\r?\n//;
  ($res) = $t=~/\bname=['"]?RAVN([A-Z]+)/;
  $t=~s!^content-.+\r?\n!!gim;
  $res{$res} = $t;
};

$GOLDflag=0;
for (split /\n/, $res{'LEX'}) {
  $GOLDflag ||= /^ORTO:/i;
  ($o5,$p5) = $GOLDflag?
     /^ORTO:(\S+)\tPPOS:\S*\tPHON:(\S+)/i:
     /^([^#]\S*)\s+\[(.*?)\]/;
  $o2p{ lc $o5 }=$p5 if $o5;
};
($rulelimit) = $res{'RULELIMIT'}=~/^(\d+)\r?$/m; $rulelimit||=$lim;

# ------------ parse dic and textgrids ------------

$lines = $res{'TEXTGRID'};

for (split /\n+/,$lines) {
  /^\s+name\s*=.+(SAMPA|orthography|SPK0\|t\|v)/i and $tier=$1 and next;
  /^xmax\s*=\s*([\d\.]+) / and $tix=$1 and next;
  /^ {6,}xmin\s*=\s*([\d\.]+) / and $xmin=$1 and next;
  /^ {6,}xmax\s*=\s*([\d\.]+) / and $xmax=$1 and next;
  /^ {6,}text\s*=\s*"(\S.*) " \r?$/ and
    $seg{$tix}{"$xmin-$xmax"}[$tier eq 'SAMPA'? 1: 0]=$1 and ++$cseg;
};

# ------------ main ------------

for $name (keys %seg) { for $xm (sort keys %{$seg{$name}}) {
  ($oseg,$pseg) = ( $seg{$name}{$xm}[0], $seg{$name}{$xm}[1] );

  $pseg=&normp($pseg);

  next if $oseg!~/\S/ or $pseg!~/\S/; $csyncseg++;
  $pseg=~s/[ ":]//g;  # remove diacritics and spurious space chars from p-form
  $flag=0; $plex='';
  for $o (split /\s+/,$oseg) {
    $p = &normp($o2p{lc $o}); if ($p=~/\S/) {$plex.=$p} else {$flag=1};
  };
  next if $flag or $plex!~/\S/; # die "$flag\t$plex\t$pseg\n";
```

```
  ($fit_,$xA_,$xB_) = &alignp($plex,$pseg);
  @xA=@{$xA_}; @xB=@{$xB_}; @fit=@{$fit_};
  &printfit($fit_,$xA_,$xB_,$plex,$pseg);
  for (0..$#xA) {$mis{"@{$xA[$_]} -> @{$xB[$_]}"}++; $cmis++};
  $cfit+= scalar @fit;
}};

for (sort keys %mis) {push @c,"$mis{$_}\t($_)\n"};
for (sort {$b<=>$a} @c) {
  my($c); ($c)=/^(\d+)/;
  $c>0 and ++$RC;
  $c>=$rulelimit and ++$RCq and $OUTmis .= "$_"
};

# ---------------------- write output --------------------

print <<RAVNRAVN
Content-type: text/html

<html>
<head>
</head>

<body bgcolor=#eeeebb>

<h1>RAVN pronunciation variation mapper</h1>

<hr>
<b>Compares lexical and descriptive phonetic forms</b>,<br>
reporting results as (i) frequency-sorted
transduction rules SAMPA&lt;=&gt;SAMPA, (ii) word-by-word analyses.

<hr>
<form
  action="https://yourserver.xyz/cgi-bin/Ravn/MakePhoneClusters_1_0.cgi"
  method=POST enctype="multipart/form-data">
  <p>

  Lexicon ('fuldformsordbog' or 'guldordbog') as .csv <br>
 <input type="file" name="RAVNLEX" accept=".csv">

<p>

  Transcription data as .textGrid
    (or a concatenation of several .textGrid<i>s</i>) <br>
  <input type="file" name="RAVNTEXTGRID" accept=".textGrid">

<p>
  Transduction rules reported must have at least
    <input type="text" size=2 value=$rulelimit name="RAVNRULELIMIT">
  instancies<br>
  <input value="Upload resources" type=submit>

</form>

<hr>

<b>Transduction rules (total=$RC, in range=$RCq, parsed-as-gold=$GOLDflag)</b>
<pre>$OUTmis</pre><hr>
<b>Lexeme analyses</b>
<pre>$OUT</pre><hr>

</body>
```

```
</html>

RAVNRAVN
;

# ------------ subs ------------

sub normp {$_=$_[0]; s/[ ":]//g; $_};   # normalize phons for comparison

sub alignp {    # NB! both args must be length>0
  my(@a,@b);
   @a=split '',$_[0]; @b=split '',$_[1];
  my(@fit,@xA,@xB,$i,$j); $i=$j=0;

  for (;;) {
    if ($i<=$#a and $j<=$#b and $a[$i] eq $b[$j]) {
      push(@fit,$a[$i]); $i++; $j++
    }
    elsif ($i<$#a and $j<$#b and $a[$i].$a[$i+1] eq $b[$j+1].$b[$j]) {
      push(@xA,[@a[$i,$i+1]]); push(@xB,[@b[$j,$j+1]]); $i+=2; $j+=2
    }
    elsif ($i<$#a and $j<=$#b and $a[$i+1] eq $b[$j]) {
      push(@xA,[@a[$i]]); push(@xB,[]); $i++
    }
    elsif ($i<=$#a and $j<$#b and $a[$i] eq $b[$j+1]) {
      push(@xA,[]); push(@xB,[@b[$j]]); $j++
    }
    elsif ($i<$#a and $j<$#b and $a[$i+1] eq $b[$j+1]) {
      push(@xA,[@a[$i]]); push(@xB,[@b[$j]]); $i++; $j++
    }
    elsif ($i<$#a-1 and $j<=$#b and $a[$i+2] eq $b[$j]) {
      push(@xA,[@a[$i,$i+1]]); push(@xB,[]); $i+=2
    }
    elsif ($i<=$#a and $j<$#b-1 and $a[$i] eq $b[$j+2]) {
      push(@xA,[]); push(@xB,[@b[$j,$j+1]]); $j+=2
    }
    elsif ($i<$#a and $j<$#b-1 and $a[$i+1] eq $b[$j+2]) {
      push(@xA,[@a[$i]]); push(@xB,[@b[$j,$j+1]]); $j+=2; $i++
    }
    elsif ($i<$#a-1 and $j<$#b and $a[$i+2] eq $b[$j+1]) {
      push(@xA,[@a[$i,$i+1]]); push(@xB,[@b[$j]]); $i+=2; $j++
    }
    elsif ($i<$#a-2 and $j<=$#b and $a[$i+3] eq $b[$j]) {
      push(@xA,[@a[$i..$i+2]]); push(@xB,[]); $i+=3
    }
    elsif ($i<=$#a and $j<$#b-2 and $a[$i] eq $b[$j+3]) {
      push(@xA,[]); push(@xB,[@b[$j..$j+2]]); $j+=3
    }
    elsif ($i<$#a-2 and $j<$#b and $a[$i+3] eq $b[$j+1]) {
      push(@xA,[@a[$i..$i+2]]); push(@xB,[@b[$j]]); $i+=3; $j++
    }

    elsif ($j<$#b-2 and $i<$#a and $a[$i+1] eq $b[$j+3]) {
      push(@xA,[@a[$i]]); push(@xB,[@b[$j..$j+2]]); $j+=3; $i++
    }
    elsif ($i<=$#a and $j<=$#b) {
      push(@xA,[@a[$i]]); push(@xB,[@b[$j]]); $i++; $j++
    }
    else {
      push(@xA,[@a[$i..$#a]]) and push(@xB,[]) if $i<=$#a;
      push(@xB,[@b[$j..$#b]]) and push(@xA,[]) if $j<=$#b;
      last
    };
  };
```

```
([@fit],[@xA],[@xB])
};

sub printfit {
  my($fit_,$xA_,$xB_,$plex,$pseg) =  @_; my(@xA,@xB,@fit,$mis);
  @xA=@{$xA_}; @xB=@{$xB_}; @fit=@{$fit_};
    die "$#xA!=$#xB!" unless  $#xA==$#xB;
  $mis="";
  for (0..$#xA) {
    $mis.="(".join("-",@{$xA[$_]})." -> ".join("-",@{$xB[$_]}).") "
  }
  $OUT .= "[$plex]\t[$pseg]\tFIT=@fit\t$mis\n";

};

# ------------ the end ------------
```

9.3 MakeWdList

```
#!/usr/bin/perl
# Program MakeWdList_1_0.cgi
# Distribution license CC-BY (RAVNUR project group, https://www.ravn.fo)

# ---------------- SETTINGS ------------------

$RemoveFromPhon = "\"\'\!\%\~";
$GoodGraphs = 'a-zA-ZáýðøúóíæÁÐÍÓÚÝÆØ_';
$GoodPhones = ':023456789aAbcCdDeEfghHiIjJklLmMnNoOpqQrRsStTuUvwWxXyYzZ';
$KeepTogether = 'AJW:';
@PoS = ('N','V','Adj','Adv','Num','Prep','Pron','Konj','Art','Int');
%PoS = map( ($_=>1), @PoS );

# ($clusterRE,$clusterS) = ( ' ([AJW:])', '\\$1' );

$AtomsForPrint = '['.join('][',@TreatAsAtom).']';
for (@TreatAsAtom) {s/[$RemoveFromPhon]//g; $AtomRE.="$_|"}; chop($AtomRE);

# -------------- INPUT ---------------

read(STDIN,$datax,%ENV{'CONTENT_LENGTH'});
($minl,$maxl) = (4,12);

$minl = $1 if $datax=~/MINL358\W+(\d+)/s;
$maxl = $1 if $datax=~/MAXL358\W+(\d+)/s;

# -------------- MAIN ---------------

$flag=0; $GOLDflag=0;

for (split /\n+/,$datax) {
  $GOLDflag ||= /^ORTO:/;
  last if $flag and /WebKitFormBoundary/i;
  next if /^-/ or /^Content-/i or !/\S/;
  ($o,$p) =
    $GOLDflag? /^ORTO:(\S{$minl,$maxl})\t+PPOS:\S*\tPHON:(\S+)/:
               /^(\S{$minl,$maxl})\s.*?\[([^\[\]\s]+)\][^\[\]]*$/;
  next unless $o;                              # words with single phform only
  $p =~ s/[$RemoveFromPhon]//g;
  $ent = "$o\t[$p]\n";
  $rejected .= $ent and next
    if $o!~/^[$GoodGraphs]+$/ or $p!~/^[$GoodPhones]+$/;
  $p = join ' ',split('',$p); $p =~ s/ ([$KeepTogether])/$1/g; # dipht handling
  %p=(); for ($p=~/(\S+)/g) {$p{$_}=1; $ptot{$_}=1};
  push(@buf, "$o\t[".join(' ',sort keys %p)."]" );
  $validc++;  # valid forms count
  $flag=1;
};

@buf = map( /^\S+\t(.+)/g , sort map(rand."\t$_",@buf) );  # Randomize




while (%ptot) {
  $maxcover=0;
  for $ent (@buf) {
    ($o,$p) = $ent=~/^(\S+)\t\[(.*)\]/;
```

```
    $cover = 0; for ($p=~/(\S+)/g) {$cover += $ptot{$_}};
    if ($cover>$maxcover) {$maxcover=$cover; $maxp = $p; $maxo=$o};
  };
  $iter++;
  $wdlist .= "<tr><td>#$iter</td>".
             "<td>$maxo</td><td><code>$maxp</code></td><td>$maxcover</td>".
             "<td><code>".join(' ',sort keys %ptot)."</code></td></tr>\n";
  push @wds0, length($maxo)."\t$maxo";
  for ($maxp=~/(\S+)/g) {delete $ptot{$_}};
  last unless $maxcover;
};

$wds = join "<br>\n",map(/\t(\S+)/g ,sort {$a<=>$b} @wds0);

# -------------- OUTPUT ----------------

$txt1 = <<RAVNEMOR
<hr>
<form action="https://yourserver.xyz/cgi-bin/Ravn/MakeWdList_1_2.cgi"
      method=POST enctype="multipart/form-data">
  <p>
  Word length (min and max):
  <input size=2 value="$minl" name="MINL358" type=text>
  -
  <input size=2 value="$maxl" name="MAXL358" type=text>

 <p>

  <input id="filhejs" value="Din fil" type="file" name="dinfil" accept=".csv">

  <p>
  <input value="Upload lexicon" name="dit ord" type=submit>
</form>

<b>OBS! Large lexicons make take a while to upload...</b>
RAVNEMOR
;

$txt2 = <<RAVNEFAR
<hr>
<b>READING LIST (count $iter, length $minl-$maxl, sorting short-long,
  parsed-as-gold: $GOLDflag)<p>$wds</b>
<hr>
<p>
<b>PROCESS LOG (word length $minl-$maxl):</b>
<p>
<table border=1>
<tr><th>iter</th><th>word</th><th>phones</th><th>cover</th><th>residual</th></tr>
$wdlist
</table>

<hr>
<b>GOOD ENTRIES (count): $validc</b><br>
<b>BAD ENTRIES (listing):</b><br><pre>$rejected</pre>
RAVNEFAR
;

$formsection = $validc? $txt2: $txt1;

print <<RAVNRAVN
Content-type: text/html

<html>
<head>
```

```
</head>

<body bgcolor=#ccffff>

<h1>Word list generator</h1>
<i>Updated 17.3.2021</i>
<hr>

<pre>
IGNORED PHONE SYMBOLS:  [$RemoveFromPhon]
ACCEPTED PHONE SYMBOLS: [$GoodPhones]
TREATED AS POSTFIXED:   [$KeepTogether]
ACCEPTED GRAPHEMES:     [$GoodGraphs]
</pre>

<p>
$formsection
<hr>
</body>
</html>

RAVNRAVN
;
```

9.4 MakeLemma

This and that – here comes the code – and I say it's allright

---

```
#!/usr/bin/perl
# Program MakeLemma_1_0.cgi
# Distribution license CC-BY (RAVNUR project group, https://www.ravn.fo)

# ----------------- SETTINGS ------------------

$GoodGraphs = 'a-zA-ZáýðøúóíæÁÐÍÓÚÝÆØÅüé\.\-\'\_0123456789';
$GoodPoS = 'A-Z1-3\[\]\.\=\-';
$GoodPhon = '234589ACDEHIJLMNORSUWXYZabdefghijklmnoprstuvwxyz\!\:\~\%';

$url = "https://myservername.dk/cgi-bin/MakeLemma_1_6e.cgi";
$priobonus = 1000;    # 'prioritized' forms are increased by $priobonus

# -------------- INPUT ----------------

($Q = $ENV{'QUERY_STRING'}) =~ s/\%(..)/pack('C',hex($1))/eg;
($TIME) = $Q=~/\bT=(\d+)/;

read(STDIN,$datax,$ENV{'CONTENT_LENGTH'});
($bound) = $datax=~/^(--+\S+)/m;

for (split /$bound\r?\n?/,$datax) {
  /\bRAVNORTO\b\W*?([$GoodGraphs]+)/s and $ortoI=$1 and next;
  /\bRAVNPPOS\b\W*?([$GoodPoS]+)/ and $pposI=$1 and next;
  /\bRAVNPHON\b\W*?([$GoodPhon]+)/ and $phonI=$1 and next;
  /\bRAVNPRIO\b\W*?([;$GoodGraphs]+)/
    and $ortoPRIO=' '.join(' ',split(/;+/,$1)).' ' and next;

  /\bRAVNCLEM\b\W*?(\d+)/ and $clem=$1 and next;

  /\bRAVNOBSOLETE\b\W*?(yes|no)/ and $obsolete=$1 and next;
  /\bRAVNFOLD\b\W*?(yes|no)/ and $dofold=$1 and next;
  /\bRAVNRECYCDICT\b\W*?(yes|no)/ and $recycdict=$1 and next;

  $t=$_;  $t=~s/^\r?\n//; $t=~s/^\r?\n//;
  ($res) = $t=~/\bname=['"]?RAVN([A-Z]+)/;
  $t=~s!^content-.+\r?\n!!gim;
  if ( $res eq 'GOLDDICT' ) { $dict = $t };
};

if ($recycdict eq 'yes') {
  $res{'GOLDDICT'} = `cat /tmp/RAVN762_GOLDDICT`;
} elsif ($dict=~/\S/) {
  $res{'GOLDDICT'}=$dict;
  open(RES,">/tmp/RAVN762_GOLDDICT"); print RES $dict; close(RES);
}

$clem||=3;
$focuswd = "$ortoI,$pposI,$phonI";
$error.="<li>Bad input ($focuswd)</li>\n" unless $ortoI and $pposI and $phonI;
$prioMin=2;


# -------------- Compose lemma suggestions ----------------

$ORTO=$PPOS=$PHON=''; $ortoF=$pposF=$phonF=''; $lexflag=$lemmaflag=0;
```

```
$REp = ' '.join(' ',&unfold($pposI));

for my $lin (split /\n/,$res{'GOLDDICT'}) {
  next unless $lin=~/^ORTO:/ or $lin=~/^---/;    # NB! not much error control!
  next if $obsolete ne 'yes' and $lin=~/\tPPOS:\S+O\t/; # ignore obsolete forms
  chomp($rawline=$lin);
  if ($lin=~/^---/) {      # lemma separator: time to process lemma
    if ($pposF) {          # does lemma's PoS conform?
      ($comInLexO,$comInLexP) = (&comX("$ortoF $ortoI"),&comX("$phonF $phonI"));
      $prio = &prio($comInLexO,$comInLexP,$ortoF,$ortoI,$phonF,$phonI);
      if ($prio>=$prioMin) {  # is O/P similarity within limits?

        ($comO,$comP) = ( &com(join ' ',@ORTO), &com(join ' ',@PHON) );
        ($dLexO=$ortoF) =~ s/$comInLexO$//; ($dLexP=$phonF) =~ s/$comInLexP$//;
        ($dInO=$ortoI)  =~ s/$comInLexO$//; ($dInP=$phonI)  =~ s/$comInLexP$//;
        push(@lemma,
              [$prio,$Fix,[@ORTO],[@PPOS],[@PHON],$dInO,$dInP,$dLexO,$dLexP] )
          if $comO=~/^$dLexO/ and $comP=~/^$dLexP/;      # is O/P map possible?
      }
    };
    @ORTO=@PPOS=@PHON=(); $ortoF=$pposF=$phonF=$Fix='';

  } else {                          # word form: store and continue
    @lin = split /\t/,$lin;
    ($orto) = $lin[0]=~/^ORTO:([$GoodGraphs]+)$/
      or $error.="<li>Bad ORTO in lex ($rawline)</li>\n";
    ($ppos) = $lin[1]=~/^PPOS:([$GoodPoS]+)$/
      or $error.="<li>Bad PPOS in lex ($rawline)</li>\n";
    ($phon) = $lin[2]=~/^PHON:([$GoodPhon]+)$/
      or $error.="<li>Bad PHON in lex ($rawline)</li>\n";
    $lexflag=1;
    push @ORTO,$orto; push @PPOS,$ppos; push @PHON,$phon;
    ($ortoF,$pposF,$phonF,$Fix) = ($orto,$ppos,$phon,$#PPOS) if $REp=~/ $ppos/;
  };
};

# -------------- select lemma suggestions, compose message ----------------

$error .= "<li>No lexical entries found</li>\n" unless $lexflag;
$error .= "<li>No relevant lemmas found in lex</li>\n" unless scalar(@lemma);

$lem_shown=0;









for my $lix ( map( \@{$_}, sort {@{$b}[0] <=> @{$a}[0]} @lemma) ) {
  ($prio,$Fix,$refO,$refPoS,$refP,$dInO,$dInP,$dLexO,$dLexP) = @{$lix};
  @ORTO=@{$refO}; @PPOS=@{$refPoS}; @PHON=@{$refP};
  @out=(); $olem=$ORTO[0]; $plem=$PHON[0];
  for my $ix (0..$#ORTO) {
    $ORTO[$ix] =~ s/^$dLexO/$dInO/; $PHON[$ix] =~ s/^$dLexP/$dInP/;
    $PPOS[$ix] = $pposI if $ix==$Fix and length($PPOS[$ix])<length($pposI);
```

```
    $PPOS[$ix] =~ s/\[(\S*?)\]/ '['.join('',sort &uniq(split '',$1)).']' /eg;
    push(@out,"ORTO:$ORTO[$ix]\tPPOS:$PPOS[$ix]\tPHON:$PHON[$ix]");
  };
  @out = &foldLemma(@out) if $dofold eq 'yes';
  $out[$Fix] = "<b>$out[$Fix]</b>";
  next if $seen{$out = join("\n",@out)}++; $lem_shown++;
  $OUT .= "--- LEMMA:".++$lemix." FIT=$prio ".
          "($olem,".&com(join(" ",@PPOS))."+,$plem) ---\n$out\n";

  ($tabout=$out) =~ s!\n+!</td></tr>\n<tr><td>\n!g; $tabout=~s!\t!</td><td>!g;
  $tableix++;
  $TABOUT .= "<tr><td align=center><b>LEMMA $tableix</b></td></tr>\n
              <tr><td>$tabout</td></tr>\n";

  last if ++$cout >= $clem;
};

$OUT .= "---\n";
$TABOUT = "<table border=1 cellpadding=4 cellspacing=0>\n$TABOUT\n</table>\n";

$error .= "<li>Last lemma is incomplete (remember final '---'!)</li>\n"
  if $ORTO or $PPOS or $PHON;

$errors = "<h2>Errors found</h2><ul>$error</ul><br><hr>" if $error and $TIME;

$lem_found = 0 + scalar @lemma;
$stat = "<i>Input</i>:<code> '$ortoI' |$pposI| [$phonI]</code><br>".
        ($ortoPRIO? "<i>Prio.</i>:<code>$ortoPRIO</code><br>": '').
        "<i>Lemmas</i>: $lem_found found, $lem_shown shown<hr>" if $TIME;

# -------------- compose <form> elements ----------------

$T = time; $c='checked';
($obs1,$obs2) =
  $obsolete eq 'yes'? ($c,''): $obsolete eq 'no'? ('',$c): ('','');
($dofold1,$dofold2) =
  $dofold eq 'yes'? ($c,''): $dofold eq 'no'? ('',$c): ('','');
($recyc1,$recyc2) =
  $recycdict eq 'yes'? ($c,''): $recycdict eq 'no'? ('',$c): ('','');

$htmlform = <<RAVNEKROG

<form action="$url?T=$T" method="POST" enctype="multipart/form-data">

  Dictionary: <input type="file" name="RAVNGOLDDICT" accept=".csv">
  <p>

  ORTO: <input type="text" name="RAVNORTO" size=20><br>
  PPOS: <input type="text" name="RAVNPPOS" size=12><br>
  PHON: <input type="text" name="RAVNPHON" size=20><p>
  PRIO: <input type="text" name="RAVNPRIO" size=20><p>


  Lemmas in output:
    <i>max.</i><input type="text" name="RAVNCLEM" size=1 value="$clem">

  <p>
  Use Obsolete?
    <i>yes</i><input type="radio" name="RAVNOBSOLETE" value="yes" $obs1>
    <i>no</i><input type="radio" name="RAVNOBSOLETE" value="no" $obs2>

  <br>
  Fold PoS?    
    <i>yes</i><input type="radio" name="RAVNFOLD" value="yes" $dofold1>
```

```
    <i>no</i><input type="radio" name="RAVNFOLD" value="no" $dofold2>

  <br>
  Recycle Dict?
    <i>yes</i><input type="radio" name="RAVNRECYCDICT" value="yes" $recyc1>
    <i>no</i><input type="radio" name="RAVNRECYCDICT" value="no" $recyc2>

  <p>
  <input value="Make lemma" type=submit>

</form>
RAVNEKROG
;

# ---------------------- output --------------------

print <<RAVNRAVN
Content-type: text/html

<html>
<head>
</head>

<body bgcolor=#fffacc>

<h1>RAVN wordform-to-lemma</h1>
<i>Updated 2.12.2020</i>
<p>
<ol>

<li>Select a GOLD dictionary
(e.g. <code>ASU_050520_guldordbog.csv</code>)</li>

<li>Insert a lexical entry
  (e.g. '<code>drongurin</code>', '<code>NCMSN==DU</code>',
        '<code>dr%ONgUrIn</code>')</li>

<li>If needed, insert prioritized lexical form(s)
  (e.g. '<code>koppurin</code>' <i>or</i>
        '<code>koppurin;risin;beiggin</code>')</li>

<li>Set options
(Use obsolete forms? Fold PoS when possible? Recycle dictionary?)</li>

<li>Press<code> <b>Make lemma</b></code></li>
</ol>

<p>

<table border=1 cellpadding=20>

<tr>
  <td valign=top>$errors  <code>$htmlform</code></td>
  <td valign=top>$stat<pre>$OUT</pre></td>
  <td valign=top><i>Lemmas as table:</i>\n<pre>$TABOUT</pre>\n</td>
</tr>
</table>

</body>
</html>
RAVNRAVN
;

# ---------------------- subs --------------------
```

```
sub com {$_=$_[0]; s!\S+!\\1\\S*!g; s!\\1!(\\S+)!; $_[0]=~/^$_/; $1};
# e.g. in = (' abc abcde','','abx  '); out = 'ab'  (space chars OK everywhere)

sub comX {$_=$_[0]; s!\S+!\\S*?\\1!g; s!\\1!(\\S+)!; $_[0]=~/$_$/; $1};
# As &com() reversed: in = (' abc','aebc xbc ',''); out = 'bc' (space chars OK)

sub dist { " $_[0]" =~ / $_[1](\S*)/g };
# e.g. in = ('abc abcde abx','ab'); out = ('c','cde','x')

sub unfold {
  $_[0]=~/(.*?)\[([^\[\]]+)\](.*)/ or return $_[0];
  my(@x);
  for my $f ( map("$1$_$3", sort &uniq(split '',$2) ) ) {push(@x,&unfold($f))};
@x};
# e.g. in = 'VAPR-[SP][MFN][ID][ARU]-[NADG]U' - out = list of all 144 forms

sub prio {
  my($comInLexO,$comInLexP,$ortoF,$ortoI,$phonF,$phonI) = @_;

  $ortoF=~/$ortoI/ + $phonF=~/$phonI/ +
  $ortoI=~/$ortoF/ + $phonI=~/$phonF/ +
  $priobonus*$ortoPRIO=~/ $ortoF / +
  length($comInLexO) + length($comInLexP)
}

sub foldLemma {  # in: list of dict entries; out: same, but folded
my(@lem)=@_; my($ix,$x,$a1,$a2,$pos1,$pos2,$b1,$b2);
for (;;) {
  ($a1,$pos1,$b1) = $lem[$ix]   =~ /^(ORTO:\S+\tPPOS:)(\S+)(\tPHON:.+)$/;
  ($a2,$pos2,$b2) = $lem[$ix+1] =~ /^(ORTO:\S+\tPPOS:)(\S+)(\tPHON:.+)$/;
  if ( $a1 eq $a2 and $b1 eq $b2 and $x=&fold2($pos1,$pos2) ) {
  @lem=(@lem[0..$ix-1],"$a1$x$b1",@lem[$ix+2..$#lem]); $ix -= $ix>0
    } else {$ix++};
    last if $ix>=$#lem;
  };
@lem}









sub fold2 {
  my($o,$err,$c,@a,@b);
  return $_[0] if $_[1]=~/^$_[0]$/; return $_[1] if $_[0]=~/$_[1]$/;

  @a = map(chop eq '['? split '': &ustr($_), "$_[0]\[" =~/([^\[\]]+[\[\]])/g );
  @b = map(chop eq '['? split '': &ustr($_), "$_[1]\[" =~/([^\[\]]+[\[\]])/g );
  return '' if $#a!=$#b;

  for my $i (0..$#a) {
    if ($a[$i] eq $b[$i]) {$o .= length($a[$i])>1? "[$a[$i]]": $a[$i]}
    else { return '' if $err++; $o .= '['.&ustr($a[$i].$b[$i]).']' }
  };

$o};

sub uniq {my(%s,@l); for (@_) { push(@l,$_) if !$s{$_}++ }; @l };
sub ustr {$_ = join('',sort split('',$_[0])); s/(.)\1*/$1/g; $_};

# ---------------------- The end --------------------
```

9.5 PushPrompt

This and that – here comes the code – and I say it's allright

---

```
#!/usr/bin/perl
# Program PushPrompt_1_0.cgi
# Distribution license CC-BY (RAVNUR project group, https://www.ravn.fo)

#   Asks for prompt file (*.txt) from local client
#   Prompt file can be a simple txt-file with lines of text (ISO) to be read
#   Optionally, text lines can be /^TEXT\tFEAS$/, FEAS --> FEA=VAL;FEA=VAL ..
#     FEA         VAL
#     fcol        red, green, ..., black (default)
#     fsize       h1, h2, ..., h3 (default)
#     clicktext   START, ..., <smiley> (default)

# ------------------- defaults, settings and init --------------------

#    --- session defs ---

$thisver = 'PushPrompt_1_2.cgi';
$url = 'https://yourserver.xyz/cgi-bin/Ravn';
$updated = '5-2-21';
$bgcol = 'ddeeff';
$Ravn_stamp = 'RavnPush12_';

#    --- prompt defs (defaults) ---

$initclicktext = 'CLICK TO START';
$finalclicktext = 'CLICK TO END';
$redotext = "<p style='color:red'>SAY IT AGAIN:</p>";

$hellotext = '';    # hello to reader ('no text' is OK)
$clicktext = '<h1>&#128522;</h1>'; # OK button (in pfile: ..;clicktext=[GO];..)
$failtext = '<h1>&#128540</h1>';  # err button (in pfile: ..;clicktext=[GO];..)
$byetext = '<h1>&#128526;</h1>';    # goodbye to reader
$fsize = 'h3';             # fontsize for prompts (in pfile: ..;fsize=h2;..)
$fcol = 'black';           # fontsize for prompts (in pfile: ..;fcol=red;..)

# ------------------- parse input --------------------

($Q = $ENV{'QUERY_STRING'}) =~ s/\%(..)/pack('C',hex($1))/eg;
($PIX) = $Q=~/\bPIX=(\d+)/;
($PPROD) = $Q=~/\bPPROD=(\d+)/;
($PTBEG) = $Q=~/\bPTBEG=(\d+\.\d+)/;
($filename) = $Q=~/\bFILENAME=([\w\.]+)/;
($STATUS) = $Q=~/\bSTATUS=(\w+)/;
$Q=~/\bCLICKTEXT=([^&]+)/ and $clicktext=$1;
$Q=~/\bFMETA=([^&]+)/ and $fmeta=$1;

$textcomment = $Q=~/\bSTATUS=fail\b/? $redotext: '';

read(STDIN,$D,$ENV{'CONTENT_LENGTH'});
($bound) = $D=~/^(--+\S+)/m;
$D=~/\bfilename=['"]([^'"]*)['"]/i and $filename=$1;
for (split /$bound-*/,$D) {
  ($name)=/\bname=['"](.*?)['"]/i;
  $content=$_;
  $content=~s/^Content.+//mig;
  $content=~s!\s*\n\s*!\n!g; $content=~s/^\n|\n$//g;
```

```
  $D{$name}=$content;
};

$PFILE = $D{'PFILE'}=~/(\d+_\d+)/? $1: $Q=~/\bPFILE=(\d+_\d+)/? $1: '';

if ($D{'PTXT'}=~/\S/) {
  @rawprom = split(/\n+/,$D{'PTXT'}); $PFILE = "".time."_$$";
  open(P,">/tmp/${Ravn_stamp}prompts_$PFILE.txt"); print P join("\n",@rawprom);
  close(P);
} elsif ($PFILE) {
  open(P,"/tmp/${Ravn_stamp}prompts_$PFILE.txt") and @rawprom = <P>;
};

@prompts = ($hellotext);
@fea = ("fcol=red;clicktext=$initclicktext");
for (@rawprom) { chomp; ($p,$f)=split /\t+/; push(@prompts,$p); push(@fea,$f) };
push @prompts,$byetext; push @fea,"fcol=red;clicktext=$finalclicktext";

# ------------------- write output ---------------------

if ($PFILE and $PTBEG and $PPROD) {
  open(P,">>/tmp/${Ravn_stamp}productions_$PFILE.txt");
  print P "$filename\t".&rnd($PTBEG,3).
    "\t$STATUS\t$PPROD\t$prompts[$PPROD]\t[$fmeta]\n";
  close(P);
};
$fmeta='';

  # $Dx = "<hr><pre>HERFRA:$D:HERTIL</pre><p><hr>\n";

if ($STATUS eq 'EXIT') {
  open(P,"/tmp/${Ravn_stamp}productions_$PFILE.txt");
  my $txt = join '',<P>;

  print "Content-type: text/plain\n\n$txt";
}
else {
  print
    "Content-type: text/html\n\n<html>\n<head></head>\n".
    "<body bgcolor=#$bgcol><h1>RAVN reading session guide</h1>$Dx\n".
    "<p>".
      ($#rawprom<0?
        &initpage: $PIX<=$#prompts?
        &promptpage($PIX): &exitpage).
     "\n</body>\n".
  "</html>";
}









# ------------------- subs -----------------

sub rnd {sprintf "%.$_[1]"."f", $_[0]};

sub initpage {
  use Time::HiRes qw(gettimeofday); my $T = gettimeofday;
```

```
  <<RAVENCLAW
    <i>Updated $updated - time now: $T</i><hr><p>
    <hr>
    <form action='$url/$thisver?PIX=0&PFILE=$PFILE' method=POST
      enctype='multipart/form-data'>
  Insert PROMPTFILE (ISO txt): <input type='file' name='PTXT' accept='.txt'>
   <p>
  or insert a valid file id: <input type='text' size=20 name='PFILE'>
   <p>
   <input type='submit' value='BEGIN READING SESSION'><p><hr>
  </form>
RAVENCLAW
};

sub promptpage {
  my($this,$next,$prev); $this=$next=$prev=$_[0];  $next++; $prev -= $prev>0;

  use Time::HiRes qw(gettimeofday);
  $TBEG = gettimeofday;

  my($urlfea) =
    "$thisver?PFILE=$PFILE&PTBEG=$TBEG&FILENAME=$filename&PPROD=$this";

  for (split /;/,$fea[$this]) {/(\w+)=(.+)/ and $$1=$2};

  <<RAVNUR
  <table border=1 style='width:100%'>
   <tr adjust=center>

    <th style='width:100'>
      <a href='$urlfea&PIX=$next&STATUS=OK&FMETA=$fmeta'><h2>
        $clicktext</h2></a>($this of $#prompts)</th>
      <th>$textcomment
        <$fsize><p style='color:$fcol'>$prompts[$this]</p></$fsize>
    </th>
  </tr>
  </table>
  <p>
<a href='$urlfea&PIX=$this&STATUS=fail&FMETA=$fmeta'>$failtext</a>
RAVNUR
};

sub exitpage {
  use Time::HiRes qw(gettimeofday); my $T = gettimeofday;
  open(P,"/tmp/${Ravn_stamp}productions_$PFILE.txt");
  my $txt = join '',<P>;

  <<STORK
  <hr>
  <a href='$thisver?STATUS=EXIT&PFILE=$PFILE'>DOWNLOAD DATA AS TABLE</a>
  <pre>$txt</pre>
  <hr>
STORK
};

# ------------------- the end ---------------------
```

9.6 EvalBlark

This and that – here comes the code – and I say it's allright

---

```
#!/usr/bin/perl
# Program EvalBlark_1_0.cgi
# Distribution license CC-BY (RAVNUR project group, https://www.ravn.fo)

# ---------------- SETTINGS ------------------

$ABC = 'a-zA-ZáýðøúóíæÁÐÍÓÚÝÆØéåÅ';
$punc = '\-\_\.'."\\'";
$GoodGraphs = $ABC.$punc;

# @PoS = ('N','V','Adj','Adv','Num','Prep','Pron','Konj','Art','Int');
# %PoS = map( ($_=>1), @PoS );
# $PoSlist = "{".join(",",@PoS)."}";

# -------------- INPUT ---------------

read(STDIN,$datax,%ENV{'CONTENT_LENGTH'});

($bound) = $datax=~/^(--+\S+)/;

for (split /$bound\r?\n?/,$datax) {
  $t=$_;  $t=~s/^\r?\n//; $t=~s/^\r?\n//;
  ($res) = $t=~/\bname=['"]?RAVN([A-Z]+)/;
  $t=~s!^content-.+\r?\n!!gim;
  if ( ($resX) = $res=~/^CHECK([A-Z]+)/ ) {
    $res{$resX} = `cat /tmp/RAVN4321_$resX`;
    $sampaOK="SAMPA er fundet (".length($res{'SAMPA'}).")" if $resX eq 'SAMPA';
  } elsif ($t=~/\S/ and $res) {
    $res{$res}=$t; open(RES,">/tmp/RAVN4321_$res"); print RES $t; close(RES);
  };
};

for ($res{'PARTOFSPEECH'}=~/(\S+)/g) {$PoS{$_}++};

# -------------- MAIN --------------

#   *********** SAMPA defined **************'

@sampa = sort {length($b)<=>length($a)} $res{'SAMPA'}=~/(\S+)/g;

#   ********** LEX checked ***********

for $lin (split /\r*\n+/,$res{'LEX'}) {
  $entc++;
  next if $lin=~/^[\s#]/ or $lin=~/^---/ or $lin!~/\S/;   # comment/separ/blank
  @f0 = split /\t/,$lin;
  $gold = $lin=~/^ORTO:/;

  if ($gold) {
    $badgold = $lin;
    $badgold =~ s/^ORTO:\S+\tPPOS:\S+(\t+PHON:\S+)+//;  # oblig. part
    $badgold =~ s/\t+COMM:[^\t]*//g;                    # optional comments
    $badgold =~ s/\t//g;    # <tab> is permittet - any remainder is non-gold!
    @f=();
    ($f[0] = $f0[0]) =~ s/^ORTO://;
    ($f[2] = $f0[1]) =~ s/^PPOS://;
```

```
    for (2..$#f0) { $f0[$_] =~ s/^PHON:(.*)// and $f[1].="[$1]" }
  } elsif ($f0[1]=~/^\[.*?\]/) {  # line format as in IS_170719_ordbog.csv
    @f = @f0;
  } else {  # line format as in RAVN1234_LEX
    @f = $f0[3]=~/\S/?
     ($f0[0],"[$f0[2]][$f0[3]]",$f0[1]):
     ($f0[0],"[$f0[2]]",$f0[1]);
  };

  $f = join '',@f[0..2]; $lexforms{ $f[0] }++;

  if (length $badgold) {
    $errtab .= &tabfy( $entc, '<nobr>Unpure</nobr>',
                       "c(".ord($badgold).")&#8658;$badgold", $lin );
  };

  if ($c = $f=~s/(\s)/$1/g) {
    $errtab .= &tabfy( $entc, 'Whitespace', $c, $lin );
  };

  if (@e = $f[0]=~/([^$GoodGraphs])/g) {
    $errtab .=
      &tabfy( $entc, 'Bad character', '{'.join(',',@e).'}', "\"$f[0]\"");
  };

  $_=$f[1];
  while (s/\[(\S*?)\]//) {
    $p=$1; $o2p{lc($f[0])} ||= $p;
    if ($p!~/\S/) { $errtab .= &tabfy( $entc, 'Empty ph-form', '[]', $f[1]) }
    elsif (scalar(@sampa)>1) {
      for (@sampa) {$p=~s!$_!!g and $seenphone{$_}++};
      $errtab .= &tabfy( $entc, 'Unknown phones', "[$p]", $f[1] ) if $p=~/\S/;
    };
  };
  $errtab .= &tabfy( $entc, 'ph-garbage', $_, $f[1] ) if /\S/;

  $errtab .= &tabfy( $entc, 'no PoS', '', $lin ) if $f[2]!~/\S/;
  for $pos (split /,/,$f[2]) {
    if (%PoS and !$PoS{$pos}) {
      $errtab .= &tabfy( $entc, ($pos? 'Bad':'Empty').' PoS', $pos, $lin );
    };
  };
};

#   ********** TextGrid(s) checked ***********

if ($res{'TEXTGRID'}) {
  my($txt) = $res{'TEXTGRID'}; chop($txt); chop($txt);
  my($phtier) = $txt=~/^\s+name = "\S*SAMPA\S*"(.+?)   item \[/msi;
  $phtier =~ s/\n//g; $phtier =~ s/ +intervals /\nintervals /g;
  for (split /\n/,$phtier) {
    my($int,$p) = /^intervals \[(\d+)\].*?text = "(.*)"/; $p=~s/""/"/g;
    $p0=$p;
    for (@sampa) {$p=~s!$_!!g and $seenphoneTG{$_}++}; $p=~s/ //g;
    $errtextgrids .= &tabfy( $int, 'Unknown phones', "[$p]", "[$p0]" )
      if $p=~/\S/;
  };

  my($orttier) = $txt=~/^\s+name = "\S*orthography\S*"(.+?)   item \[/msi;
  $orttier =~ s/\n//g; $orttier =~ s/ +intervals /\nintervals /g;
  for (split /\n/,$orttier) {
    my($int,$o) = /^intervals \[(\d+)\].*?text = "(.*)"/;
    $o=~s/""/"/g; $o=~s/<[\w ]*>/ /g; $o=~s/ +/ /g;
```

```
    my($badgraphs) = '"'.(join '" "',$o=~/([^ $GoodGraphs])/g).'"';
    $errtextgrids .= &tabfy($int, 'Illegal graphemes', $badgraphs, "\"$o\"" )
      if $badgraphs=~/[^ "]/;

    for ($o=~/([$GoodGraphs]+)/g) {
      $lexforms{$_} or
      $errtextgrids .= &tabfy($int, 'Not in lexicon', "\"$_\"", "\"$o\"" )
    };
  };
}

# ------------------- MANUS checked ----------------------

if (%lexforms) {
  for $wd ($res{'MANUS'}=~/([$GoodGraphs]+)/g) {
    ($wdXpunc=$wd) =~ s/[$punc]+$//g;
    $mx{$wdXpunc}++ unless $lexforms{$wd} or $lexforms{&lowerchar($wd)}
      or $lexforms{$wdXpunc} or $lexforms{&lowerchar($wdXpunc)};
    $p=$o2p{lc($wd)};
    for (@sampa) {$p=~s!$_!!g and $seenphoneMS{$_}++}; $p=~s/ //g;
  };
  for (sort keys %mx) {$errmanus .= "   '$_' ($mx{$_})<br>\n" };
}

# ---------------------- SAMPA checked --------------------

for (@sampa) {
  $sampa0 .= "[$_]" unless $seenphone{$_};
  $sampa00 .= "[$_]" unless $seenphoneTG{$_};
  $sampa000 .= "[$_]" unless $seenphoneMS{$_};
};

# -------------- OUTPUT ----------------

$txtbody = <<RAVNEMOR
<form action="https://yourserver.xyz/cgi-bin/Ravn/EvalBlark_1_9.cgi"
      method=POST enctype="multipart/form-data">

<table cellpadding=5 border=2>
<tr>
<td>
  SAMPA (phone inventory) as .txt <br>
  Select previous <input type="checkbox" name="RAVNCHECKSAMPA" checked>
    or
  <input type="file" name="RAVNSAMPA" accept=".txt">
</td>
<td>
  PAROLE (PoS inventory) as .txt <br>
   Select previous <input type="checkbox" name="RAVNCHECKPARTOFSPEECH" checked>
  or
  <input type="file" name="RAVNPARTOFSPEECH" accept=".txt">
</td>

<td>
  Lexicon (annotated word forms; <i>GOLD</i>-standard is supported) as .csv <br>
    Select previous <input type="checkbox" name="RAVNCHECKLEX">
  or
 <input type="file" name="RAVNLEX" accept=".csv">
  <p>
</td></tr>

<tr>
<td>
  Transcription as .textGrid (ISO-8859)<br>
```

```
   Select previous <input type="checkbox" name="RAVNCHECKTEXTGRID">
  or
  <input type="file" name="RAVNTEXTGRID" accept=".textGrid">
</td>
<td>
  Manus as .txt (recommended) or .rtf (experimental)<br>
    Select previous <input type="checkbox" name="RAVNCHECKMANUS">
  or
 <input type="file" name="RAVNMANUS" accept=".txt">
</td>
<td>
  <input value="Upload resources" type=submit>
</td>
</tr>
</table>

</form>
RAVNEMOR
;

$report .= $entc==0? "<h3>No LEXICON uploaded</h3>":
          $errtab!~/\S/? "<h3>No errors found in LEXICON</h3>":
          "<p>\n<hr><b>Errors found in LEXICON</b> (in $entc entries)<p>
           <table border=1>".
          "<tr><th>line</th><th>errortype</th>".
          "<th>instance</th><th>field</th></td>$errtab\n</table>\n<hr>";

$report .= length($res{'TEXTGRID'})==0? "<h3>No TEXTGRID uploaded</h3>":
          $errtextgrids!~/\S/? "<h3>No errors found in TEXTGRID</h3>":
          "<p>\n<hr><b>Errors found in TEXTGRID</b> <p> <table border=1>".
          "<tr><th>interval</th><th>errortype</th>".
          "<th>instance</th><th>field</th></td>$errtextgrids\n".
        "</table>\n<hr>";

$report .= length($res{'MANUS'})==0? "<h3>No MANUS uploaded</h3>":
          $errmanus!~/\S/ && %lexforms?
            "<h3>All words in MANUS were recognized</h3>":
          $errmanus!~/\S/? "<h3>MANUS uploaded, but no lexicon</h3>":
          "<p>\n<hr><b>MANUS has unrecognized words:</b> <p> $errmanus\n<hr>";

if (scalar(@PoS)==0) { $report .=  "<h3>No PAROLE uploaded</h3>" };

if (scalar(@sampa)) {
  if (length($res{'LEX'})) {
    $report .= length($sampa0) && $entc>0?
     "<p>\n<b>SAMPA phones not used in LEXICON:</b> <pre>$sampa0</pre><hr>":
       $entc>0? "<b>All ".scalar(@sampa)." SAMPA entries used in LEXICON<p><br>":
      "";
  };

  $report .= length($sampa00) && length($res{'TEXTGRID'})?
   "<p>\n<b>SAMPA phones not used in TEXTGRID:</b> <pre>$sampa00</pre><hr>":
    length($res{'TEXTGRID'})? "<b>All ".scalar(@sampa).
      " SAMPA entries used in TEXTGRID<p><br>":
   "";

  if (length($res{'LEX'})) {
    $report .= length($sampa000) && length($res{'MANUS'})?
     "<p>\n<b>SAMPA phones not used in MANUS:</b> <pre>$sampa000</pre><hr>":
      length($res{'MANUS'})? "<b>All ".scalar(@sampa).
        " SAMPA entries used in TEXTGRID<p><br>":
     "";
  };
}
```

```
else { $report .=  "<h3>No SAMPA uploaded</h3>" };

print <<RAVNRAVN
Content-type: text/html

<html>
<head>
</head>

<body bgcolor=#fff7ba>

<h1>RAVN resource checker</h1>
<i>Updated 7.1.2021</i>

<hr>
<b>Evaluating lexicon, SAMPA, TextGrids and/or Manus for consistency<p>
NB! Gold-lexicons are auto-detected, but not thoroughly PoS-checked wrt. PoS-
inventory
</b>
<p>

<b>Accepted graphemes: <code>$GoodGraphs</code></b>

$txtbody

$report
</body>
<hr>
</html>

RAVNRAVN
;

# ---------------------- subs --------------------

sub tabfy { "<tr><td>".join("</td><td>",@_)."</td></tr>\n" };
sub lowerchar {  $_=lc($_[0]); tr/ÁÐÍÓÚÝ/áðíóúý/; $_};

# ---------------------- The end -------------------------
```
