BMBF

Verbmobil
Verbundvorhaben

Automatic Conversion of American Dialogue Data into VM Compatible Format

Florian Schiel
Susanne Burger

Ludwig Maximilians Universität München

Technisches Dokument 34
September 1995

September 1995

Florian Schiel
Susanne Burger

Institut für Phonetik und Sprachliche Kommunikation
Ludwig Maximilians Universität München
Schellingstr. 3/II
80799 München

Tel.: (089) 2180 - 2807
e-mail: `schiel@sun1.phonetik.uni-muenchen.de`

Gehört zum Antragsabschnitt: 14 VERBMOBIL/PHONDAT

Die vorliegende Arbeit wurde im Rahmen des Verbundvorhabens Verbmobil vom Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie (BMBF) unter dem Förderkennzeichen 01 IV 102 L/4 gefördert. Die Verantwortung für den Inhalt dieser Arbeit liegt bei dem Autor.

Inhaltsverzeichnis

Abstract

Dieses Dokument beschreibt die automatische Überführung von Dialog-Daten im sog. Carnegie Mellon University (CMU) Format in ein Verbmobil (VM) kompatibeles Format. Insbesondere wird auf Probleme infolge des 'open definition' Konzepts der CMU Verschriftungen eingegangen. Die Transformation von Nomenklatur und Struktur der Signaldaten wird detailiert beschrieben. Da dieses Dokument sowohl von deutschen als auch amerikanischen Partnern benutzt werden wird, ist es in Englisch verfaßt.

This document describes the automatic conversion of the dialogue data collected at Carnegie Mellon University (CMU) into a Verbmobil (VM) compatible format. Certain errors caused by the 'open definition' approach of the CMU system are addressed. The conversion of file names and structure is given in detail. This document is intended for the usage of German and American Partners. Therefore it is written in English.

1 Introduction

One of the tasks within Teilprojekt 14 is the automatic conversion of signal data into a VM compatible format that are not produced according the 'Handbuch zur Datenaufnahme und Transliteration in TP14 von VERBMOBIL – 3.0' ([1]). For this purpose several tools were developed at Munich University that allow the semi-automatic conversion of transliterations, the validation of the signal data and the semi-automatic conversion of file names and data structure into VM compatible formats. The following sections describe these conversions and problems that occurred and should be known by the end user of the data. Appendix A shows the man page for the conversion program cmu2vm.

2 Transformation of signal files

The following steps have to be performed for the conversion of CMU structure and file naming into VM compatible formats:

- Copy signal files from several directories to common directory
- Remove Rockridge Extension Information Files
- (Create PhonDat 1 headers – Only for data that come without PhonDat 1 headers, e.g. Multicom data)
- Change names using the tool `cmudirnames2vm.csh`
  This tool steps into each directory containing a dialog, examines the correct order of turn numbering (CMU starts with 1) and changes signal file names into VM file names. Finally the directory is renamed to the appropriate VM dialog name.
  For CMU recorded data we use the locator 'c' and the dialog type 'r' (= separated room, push button, English), e.g. `r100c`.
  The American dialogs are numbered in alphabetical order on each volume. The next volume starts with the last dialog number of the previous volume plus one.
- Verify four randomly selected dialogs by hand.
- Patch headers using the tool `patch_phondat_filename`
  Since the file naming is changed to VM format the entries in the PhonDat headers need to be updated.
- Archive Master into BAS archive

3 Validation and documentation

The standard documentation files are linked to the master. Since these data might be used by others than German partners, we provide an English translation of the documentation as well. The man page `cmu2vm` for the automatic conversion of the transliterations is linked to the master to give hints about the usage of the transliterations to the users.

The tool `test_phondats` is run over all signal files in the master (Options: `precise=1 f=1`). This tools creates a validation protocol which has to be examined by hand. The tool checks for

- header structure, size and version
- header entries (as far as they are able to be verified).
- file size.

- readability of signal data.

Known and tolerated errors are:

- The date items (DAY, MONTH, YEAR) in the PhonDat 1 header are set to 0 or to the date of conversion into VM format instead of the date of recording.
- The header item SPRK (speaker ID) is not set (because in VM three characters are used instead of two).

Other errors have to be fixed using a tool set (`patch_phondat_*`) and the validation procedure has to be re-run.

Another final validation is run to the first pressed CD-ROM to ensure that there haven't been any transmission errors from master archive to CD-ROM production line.

4 Transformation of transliterations

The following steps have to be performed for the conversion of the transliterations in CMU format into a VM compatible format:

- Copy transliteration files from several directories to common directory
- Transform transliterations using the tool `cmu2vm`
  `Cmu2vm` reads all transliteration files found in the command line and creates new files with extension `.crl` with the transformed transliteration. The process is logged into a file, which has to be examined by hand for errors. See appendix A for details of the conversion.
  If errors occur that require an extension of the tool syntax, the capability of the tools is extended and the conversion is re-run until no errors occur anymore (see below for a discussion of systematic problems).
- Remove the original transliteration files
- Rename the filenames and structure of the transliteration files to match the VM format of the signal files (tool `cmutrlnames2vm`)
  Care has to be taken that the same parameters are used as during the renaming of the signal files to yield matching structures.

- Run the VM formatter `convdlg` (University of Kiel) over all transliteration files
  This tool re-format the line spacing in the files to conform VM guidelines and produces an error log, if it encounters format errors, that cannot be resolved. These errors have to be fixed by hand and the formatter is run again until no errors are found any more.

- Produce documentation for the transliteration package (including man page `cmu2vm`)

5 Discussion of Problems

In the following we'll discuss some of the systematic problems that we encountered during our work. These problems can be divided into

- problems caused by missing markers in the CMU format
- problems caused by missing markers in the VM format
- problems caused by errors in the input data
- problems caused by the 'open definition' of the CMU format

5.1 Missing markers in the CMU format

Technical breaks are not marked in the CMU format. Consequently the transformed transliteration won't contain any markers `<T;>`.

Non-lexical items are not marked in in the CMU format. Consequently the transformed transliteration won't contain any markers `*`.

The usage of **period, comma and question mark** are not compatible to the usage in the VM format. Therefore these markers are transformed into VM comments (e.g. `{comma}  -> <;comma>`).

Items hard to understand are not marked in in the CMU format. Consequently the transformed transliteration won't contain any markers `%`.

Non understandable items are not marked in in the CMU format as an own category like in the VM format. However, single letters, e.g. `[a]` are mapped to the VM marker for non understandable items `<%>`.

Phrase break off cannot be distinguished from repeats/repairs in the CMU format. Therefore all found markers of this kind are mapped to repeats/repairs in VM format. Consequently there will be no markers `/-` in the output.

Bracketing is not a general feature of the CMU format. However, human and non-human noise markers can be used as starting and ending points for noises. The conversion is done correctly into VM bracketing in these cases. If a starting label without an ending label is found in one turn, an error will be issued.

Spellings are not marked in the CMU format except that they are put in capital letters. Therefore the word 'I' (ich) cannot be distinguished from an spelled 'I' like in 'C I A'. We solved this problem by examining the neighboring items around a single capital 'I'. If there are other single capital letters found in the left or right context the 'I' is treated as a spelling and marked accordingly (VM format: `$I`). Otherwise it is traeted as the word 'I' (ich). The only error that might occur here is a single 'I' that meant to be a spelling, e.g. '...my name is Inuoe with an I at the beginning...'

5.2 Missing markers in the VM format

Semantic Markers are not used in the VM format. The CMU marker `{seos}` is mapped into a comment `<;seos>` in the output.

5.3 Errors in the input data

No automatic conversion is thoroughly save against format errors in the input. For this reason we decided to copy the original transcription of CMU as line comments right after each turn in the output. Users that encounter strange entries can look for the original text and (hopefully) understand what was wrong in the input.

5.4 'Open definition' format

The transscription convention of CMU does not contain a closed and well defined list of all allowed markers. As a consequence every transcriber will 'invent' new markers as he pleases. Of course this cannot be handled automatically. However, in most of the cases the program will recognize a new marker and it's general type (e.g. articulatory versus non-articulatory noises) and map it to the defined rest category of the VM format. Additionally a warning is printed into the protocol.

We try to extent the syntax of the tool `cmu2vmi` after each volume, but in some cases (for instance hesitations) it's impossible to find out all the different types of markers used.

Another point are **multiple markers**, which means that one transcriber uses another marker than the others for the same effect, e.g. we found the markers `#begin_electronic_hum#` and `#begin_hum#` on different volumes.

6 Conclusion

The semi-automatic conversion of the American dialogue data turned out to be feasable with some constraints. The produced and disseminated output should be compatible to the VM format by syntatctic means. The semantics however might be distorted by the fact that the two used formats VM and CMU are semantically not equivalent. Users are urgently asked to keep in mind these problems before exploiting these data.

References:

[1] Kohler,Lex,Ptzold,Scheffers,Simpson,Thon (1994): Handbuch zur Datenaufnahme und Transliteration in TP14 von VERBMOBIL – 3.0. VERBMOBIL Techdok-11-94, University of Kiel.

Appendix A

Man Page cmu2vm

```
CMU2VM(1)                                                  CMU2VM(1)


NAME
       cmu2vm  -  converting CMU/KA transliteration files into VM
       format

SYNOPSIS
       cmu2vm help=yes
       cmu2vm [v=yes|no] file1 [file2 ...]


DESCRIPTION
       Cmu2vm opens all  files  that  are  in  the  command  line
       (transliterations  in  CMU/KA  format), reads turn by turn
       and transforms it into a VM compatible format. The  output
       for  each  transformed  file is written to a file with the
       extension ´.crl´. If the input file name had a  extension,
       it  is removed. If not, the extension ´.crl´ is just added
       to the filename.
       Note that the output is NOT formatted. That means one turn
       is written into one line without any CR. You must use a VM
       formatter (e.g. convdlg12_94) after the transformation.
       In the default usage there is no output to stdout,  except
       the  version and date of the source code. Errors found are
       reported to stderr.  If a severe error  occurs  that  pro-
       hibits  the check of further files on the command line, an
       error message is printed to stderr and the  command  exits
       with  a  negative  exit  code.   For  instance,  if a non-
       existent filename is in a row of filenames in the  command
       line.   Cmu2vm  will  skip this file and continue with the
       next in command line.  The same happens, if one turn is to
       large  to  be read in memory (very unlikely).  The correct
       exit code is 0.


OPTIONS
       Cmu2vm uses icsiargs(3) style command line options.
       v=yes|no  verbose. If yes,  prints  all  read  information
                 about  the  files  to  stdout.   Defaults  is no
                 (silent mode).

FILES
       /usr/local/bin/cmu2vm
       Sources under sun1:/home/schiel/bas/cmu2vm

SEE ALSO
       icsiargs(3)


WHAT IS DONE HERE
       Cmu2vm performs the following transformations:

       Comments  Comments (enclosed in ´{...}´) are copied to out-
                 put  as  comments (enclosed in ´<;...>´). Excep-
                 tion is ´{filled}´ which is mapped to ´<Z>´.


                                                                1





CMU2VM(1)                                               CMU2VM(1)


                 Comments of whole lines (beginning with ´;´) are
                 copied to the output.

       ´<...>´   Parts  enclosed  in  ´<...>´ (CMU) are copied to
                 parts enclosed in ´+/.../+´  (see  Comments  for
                 exception).  Therefore  there  ist no difference
                 between repair/repetition  and  sentence  break.
                 Items  that are mapped into ´<...>´ (VM) and are
                 at the end or the beginning within an repeat are
                 moved  outside  of  the bracketing. Same happens
                 with initial items. There are many special cases
                 in  the  handling  of the repeat. Please see the
```

commented source code for details.

Turnmarkers

are transformed into VM format. The speaker ids are made by ommitting the sex id in the CMU/KA speaker ids. E.g. facm -> ACM.

´@...@´ Shorted pronunciations (e.g. don´t @do not@) are transformed into the correct VM format (do not <!2 don´t>).

Wordbreaks

Word breaks (<[agai(nst)]>) are transformed into the VM word break (+/agai=/+). Care is taken that the bracketing is conform with the other brackets around. Single letters (<[a]>) are matched to unintelligible (<%>).

Mispronunciations

Mispronunciations ([firth (first)]) are transformed into a VM variant (first <!1 firth>).

Unintelligable

Unintelligables ([fr]) are transformed into <%>.

Spellings Spellings (simple capital Letters) are treated correct (VM: $ + capital letter), even in the case of ´I´. The only possible error might be a single spelling ´I´, because this cannot discriminated from an ´I´ (ich).

´/.../´ Articulatory noise is treated correct if known. Unknown markers are transformed to <Ger"ausch>. Known markers are:
/h#/ /lg/ /lt/ /begin_laugh/ /end_laugh/ /ls/ /ts/ /throat_clear/ /gulp/ /glottal/ /sneeze/ /sniff/ /snort/ /yawn/ /cg/ /eh/ /ah/ /aw/ /er/ /oh/ /oo/ /uh/ /mm/ /hm/ /nn/ /um/ /ch/ /hoo/ /mf/ /phoo/ /yah/ /ha/ /huh/ /phew/ /pf/ /choo/ /noise/ /whistle/ /sigh/ /slurp/ /grunt/ /unintelligible/

```
     Noise    Non-articulatory noise (´#...#´) is  transformed
              correct as far as

                                                             2


CMU2VM(1)                                            CMU2VM(1)


              known.  Unknown  noise is transformed into ´<#>´.
              There ´s lots of known noise in the  code.  How-
              ever  we  always  find  something  new since the
              CMU/KA format is open.

     Bracketing
              There is no bracketing  in  the  CMU/KA  format.
              However  there  are markers for begin and end of
              an event. These  are  transformed  into  correct
              bracketed  events  according  to  the VM format.
              Unknown begin or end markers are transformed BOTH
              to   ´<#>´!  Known   bracketing   markers   are
              /begin_laugh/ #begin_headset# #begin_mechanical#
              #begin_microphone#                 #begin_click#
              #begin_key_click#           #begin_paper_rustle#
              #begin_phone_ring#   #begin_tap#   #begin_noise#
              #begin_scratching#       #begin_electronic_hum#
              #begin_beep#   #begin_rustle#   #begin_static#
              #begin_hum#.
              Example:  /begin_laugh/   text   text   text
              /end_laugh/  will  be transformed into <:<Lachen>
              text text text:>

WARNINGS
     The output is NOT fully compatible  to  the  VM  standard.
     However  this  filter  tries  to produce at least a syntax
     that goes conform.  Errors are to be expected in the fol-
     lowing cases:
     Repeats/repairs  and  sentence  breaks  cannot  be distin-
     guished in the CMU data.
     Technical breaks are not marked in the CMU data.
     Non-lexical items and word breaks are  somewhat  mixed  in
```

the CMU data. We therefore do not guarantee that these are mapped correct in all cases.
Missing items:
The usage of period, comma and question mark are not compatible to the usage in the VM format. Therefore these markers are transformed into VM comments ({comma} -> <;comma>, ...).

BUGS

Shorted pronunciation that are from one word (which should be correctly then a variant, but can be found in the input) like can´ t @cannot@ result in an error.

COPYRIGHT