UAM Text Tools — a flexible NLP architecture

Tomasz Obrębski, Michał Stolarski

Adam Mickiewicz University
Umultowska 87, 61-614 Poznań, Poland
{obrebski,mstolar}@amu.edu.pl

Abstract

The paper presents a new language processing toolkit developed at Adam Mickiewicz University. Its functionality includes currently tokenization, sentence splitting, dictionary-based - morphological analysis, heuristic morphological analysis of unknown words, spelling correction, pattern search, and generation of concordances. It is organized as a collection of command-line programs, each performing one operation. The components may be connected in various ways to provide various text processing services. Also new user-defined components may be easily incorporated into the system. The toolkit is destined for processing raw (not annotated) text corpora. The system was originally intended for Polish, but its adaptation to other languages is possible.

1. Introduction

Automatic extraction of linguistic information from text corpora has for many years been an important research direction within natural language processing area. A number of computer systems and data resources have been developed. Two examples of such systems, which may be used to process Polish corpora, are Poliqarp (Przepiórkowski and et al., 2004) and Intex (Silberztein, 1993). Poliqarp is a search engine accompanying IPI PAN Corpus[1] — a large (300 million segments) corpus of Polish (Przepiórkowski, 2004). Intex is a corpus processor, which — in contrast to Poliqarp — is able to work with arbitrary text data supplied by the user.

Systems of this type are addressed to end-users: linguists or computational linguists. The functionality and the range of possible applications of such systems is fixed by their architects and difficult or impossible to extend. In the present time, applications of natural language processing technologies become more and more common, and text corpora become more and more accessible. Therefore, the need emerged for tools offering similar functionality, but addressed towards language engineering community: easily configurable, independent of language, annotation format and contents, extensible with user-defined components, and easily connectible to other software.

The project called UAM Text Tools (UTT) started at the end of 2004 as an attempt to respond to this need. UTT is meant to supply a set of basic instruments for building various applications related to text corpus processing. It is a package of language processing tools. Its functionality includes currently tokenization, sentence splitting, dictionary-based morphological analysis, heuristic morphological analysis of unknown words, spelling correction, pattern search, and generation of concordances. One more component, which is almost ready to be integrated with the package, is a dependency parser. The package is freely available for research and educational use.

2. The architecture

The system is organized following the software engineering principles adopted in UNIX environments: as a collection of independent components, each providing one language processing service (tokenization, lemmatization, search,...). The components are independent command-line programs working as filters, communicating through pipes. The unifying element of the whole is the uniform i/o file format, in which annotated corpus data is stored and exchanged between component programs.

The UTT file format was designed to meet two principal requirements:

- simplicity of processing with standard widely available text processing utilities and programming languages (grep, sed, AWK, Perl, lex),
- human readability.

Therefore, simple line-based text format with space-separated fields was chosen. Each line of a UTT-formatted file describes a continuous segment of the input text file, usually a token. There are four mandatory fields: start *position* in the input file, the *length* of the segment, the segment *type* (word-form, number, space), and the orthographic *form* of the segment. Any number of annotation fields may then follow. These fields contain annotation introduced by processing programs.

Below is the sample of the UTT file with morphological annotation.

```
0000 04 W szła  lem:i ść,V/AiVpMdTaNsP3Gf
0004 01 B _
0005 10 W dzieweczka   lem:dzieweczka,N/ChGfNs
0015 01 B _
0016 02 W do  lem:do,P
0018 01 B _
0019 08 W laseczka   lem:laseczka,N/GfNsCh
0019 08 W laseczka   lem:laseczek,N/GiNsCg
0027 01 P .
0028 01 B \n
```

Figure 1: Example of UTT file

Annotation fields are composed of the field name, in this case `lem:` , and the value, which is an arbitrary string of non-white characters. The field name in the above example is the name of the component which introduced the field (this is the default[2]).

[1] http://www.korpus.pl

[2] This is one of the reasons for using very short, usually 3-letter long, names for component programs.

Format features:

1. ambiguous annotation (ambiguous interpretation of a segment, ambiguous segmentation) may be represented
2. parallel annotation also may be represented
3. the result of independent processing of a file by two programs may be merged (using eg. the standard UNIX `sort -m` command),
4. reference to the original text is directly accessible

A component program reads a sequence of segments from input and writes another sequence of segments to output. The parameters determine which segments the program is supposed to process, which field(s) contain the input data to the program, what should be output (successfully processed segments, unsuccessfully processed segments, all). The results of processing are added as a new annotation field.
There are two ways of representing ambiguity: either the segment appears multiple times with different annotations (`laseczka` in the example in Fig. 1), or ambiguous annotation is presented as one complex value (Fig. 2).

```
...
0018  01 B _
0019  08 W laseczka   lem:laseczka,N/GfNsCn;laseczek,\
N/GiNsCg
0027  01 P .
...
```

Figure 2: Ambiguous annotation in one field

The first format is more universal. It directly corresponds to a DAG encoding of ambiguous text data, allowing for example for representing ambiguous segmentation in natural way. This format is convenient e.g. as the input to syntactic parser. The second format is more compact and — especially in case of unambiguous segmentation — allows for performing efficient search.

3. Component programs

In this section we will shortly describe the UTT components currently available.

3.1. *tok* — a tokenizer

tok is a simple program which reads a text file and identifies tokens on the basis of their orthographic form. The type of the token is printed as the *type* field. In standard configuration five types of tokens are distinguished: word (type `W`) — continuous sequence of letters, number (type `N`) — continuous sequence of digits, space (type `B`) — continuous sequence of white-space characters, punctuation mark (type `P`) — single printable characters not belonging to any of the other classes, unprintable character (type `H`) — single unprintable character (see Fig. 3)

3.2. *lem* — a lemmatizer

lem performs morphological analysis of a simple orthographic word, returning all its possible morphological annotations, disregarding the context. The result of the analysis is added as the value of the new annotation field, which

```
0000  04 W szła
0004  01 B _
0005  10 W dzieweczka
0015  01 B _
0016  02 W do
0018  01 B _
0019  08 W laseczka
0027  01 P .
0028  01 B \n
```

Figure 3: Example of *tok* output

default name is *lem*. In case of ambiguity either the segment is muliplicated or ambiguous description in the format $lemma_1$ **,** tag_1 **,** ... **,** tag_n **;** ... **;** $lemma_m$ **,** tag_1 **,** ... **,** tag_n is used as the value (cf. Fig. 2).
lem may work with a text format dictionary or with a dictionary compiled into FSA representation.

3.3. *gue* — a guesser

Morphological analysis performed by *lem* component is based on a dictionary. Therefore it is limited to the words included in that dictionary. The *gue* component is designed to assist *lem* when it fails to recognize a word form. It proposes the most likely description(s) of a word. The output is based on the information on the frequency a specific morphological description is assigned to word-forms with a given suffix and/or prefix.

3.4. *cor* — a corrector

The spelling corrector applies Kemal Oflazer's dynamic programming algorithm (Oflazer, 1996) to the FSA representation of word-forms extracted from the *lem*'s dictionary. Given an incorrect word-form it returns all word-forms present in the dictionary which edit distance is smaller than the threshold given as a parameter.

3.5. *ser* — a pattern search tool

The *ser* program is the UTT component for locating text fragments matching a pattern. The pattern is a regular expression over terms corresponding to corpus segments. Several examples of terms follow: `seg` — any segment, `word` — any word, `word(nie.+,,<N>)` — a word beginning with 'nie' and tagged as noun, `space(. *\n. *)` — a space segment containing a newline character, `lexeme(pomoc)` — a form of the lexeme 'pomoc', `tag(<N/Ca>)` — a noun in accusative.
Term arguments may be arbitrary regular expressions.
For example the following invocation of *ser*:

```
 ser -e 'tag(<ADJ>) space (word space)? lexeme(praca)'
```

locates occurrences of adjectives followed by any form of the lexeme 'praca' (work), optionally separated by another word. Matches are indicated by inserting 0-length segments at the beginning and at the end of the match.
ser performs the search by matching character-level regular expressions against fragments of UTT file using the `flex` program. The expansion of the pattern into the coresponding character-level regular expression is implemented with the use of `m4` macroprocessor. Terms used in patterns are simply `m4` macro invocations.
The special form <...> represents a partial specification of a tag (constraints on tag form) and is expanded into a

regular expression matching all tags meeting this specification (e.g. `<N/Ca>` is expanded into a regular expression matching all tags for nouns in accusative case).

The processing speed of *ser* is over 350 000 corpus segments/sec[3] (segments = visible tokens: words, punctuation marks, numbers)

```
...
180467   06  W będący  lem:b˛edący,ADJPRP/NpCnvGp,ADJPRP       /...
180473   01  S _
180474   11  W przedmiotem    lem:przedmiot,N/GiNsCi
180485   01  S _
180486   00  BOM  ser:1
180486   10  W niniejszej    lem:niniejszy,ADJ/DpNsCgdlGf
180496   01  S _
180497   05  W pracy   lem:praca,N/GfNsCd,N/GfNsCg,N/GfN         sCl
180502   00  EOM  ser:1
180502   01  P .
180503   02  S _\n
180505   09  W Równowaga    lem:równowaga,N/GfNsCh
...
```

Figure 4: Example of *ser* output

3.6. *grp* - a grep-like tool

The *grp* component is similar to *ser* with the difference that, instead of `flex`, text scanning is performed by `grep` on a slightly modified UTT file (as `grep` operates on single lines, all segments making up a sentence are merged in one line). Regular expressions passed to `grep` are generated by *grp* using the same set of `m4` macrodefinitions as *ser*.

grp atteins the speed of over 1.5 mln corpus segments/sec for arbitrarily large corpora.

The basic use of grp is the reduction of the corpus size being passed to subsequent processing phases (e.g. to *ser* which will extract the matches), by extracting sentences in which the searched expression appears or is likely to appear. For more details on *ser* and *grp*, see (Obrębski, 2006).

The complete list of components includes also: *sen* - the sentensizer, *kot* - reconstructs original text from given UTT file, *con* - a tool for displaying concordances in human-friendly format.

4. Usage examples

Below several examples are presented, showing how UTT may be used to perform different text processing tasks.

4.1. Annotation

The following sequence of commands:

```
tok  | lem  | cor  -S  lem  | lem  -I  cor  | gue  -S  lem
```

causes the input text to be processed as follows: tokenize the text (*tok*), perform morphological analysis of words (*lem*), try to correct words for which *lem* produced no description (`cor -S lem`), perform morphological analysis of words corrected by *cor* (`lem -I cor`), guess descriptions for words which still have got no annotation from *lem* (`gue -S lem`). For example, the result of processing the sentence *Piszemy gitesowe progromy.*[4] will be:

```
0000   07  W Piszemy   lem:pisa  ć,V/AiVpMdTrNpP1
0007   01  B _
0008   08  W gitesowe   gue:gitesowy,ADJ/CanvDpGafinNp
0008   08  W gitesowe   gue:gitesowy,ADJ/CanvDpGnNs
0016   01  B _
0017   08  W progromy   cor:pogromy    lem:pogrom,N/GiNpCa
0017   08  W progromy   cor:pogromy    lem:pogrom,N/GiNpCh
0017   08  W progromy   cor:pogromy    lem:pogrom,N/GiNpCv
0017   08  W progrumy   cor:programy    lem:program,N/GiNpCa
0017   08  W progrumy   cor:programy    lem:program,N/GiNpCh
0017   08  W progrumy   cor:programy    lem:program,N/GiNpCv
0025   01  P .
0026   01  B \n
```

Figure 5: Example of *lem*/*cor*/*gue* annotation

4.2. Spelling correction

The following command:

```
tok  | lem  -p  W -o  /dev/null   | cor  --one
```

prints out all spelling errors found in text, annotated with correction suggestions. For example, for the sentence *Pizsemy dobre progromy.* the following output is produced:

```
0000   07  W Pizsemy   cor:Piszemy
0014   08  W progromy   cor:pogromy;programy
```

First, dictionary-based morphological analysis is performed and successfully processed segments are discarded. The segments the `lem` failed to analyze are sent to `cor`.

4.3. Concordancer

In this example we are looking for an adjective followed by a form of the nominal lexeme 'praca' (work). The matches are displayed by the `con` component with left and right context. The command:

```
tok  | lem  --one  \
| ser  -e  "tag(<ADJ>)    space  lexeme(praca)"    \
| con  -t  -l  20  -r  15
```

produces:

```
 będący  przedmiotem    [niniejszej    pracy].   Równowaga
     W rozwa żanym  w [niniejszej    pracy]  wyładowaniu
    Rezultaty   wy żej [wspomnianych    prac]  są w pełni
   jest  przedmiotem    [tej  pracy].   Metody
     do  zrozumienia    [niniejszej    pracy].   Profil
Z tego  powodu  autor   [niniejszej    pracy]  wraz  ze
jednej  z nast˛epnych   [swoich  prac]  Devoto  (Devoto
  Z punktu  widzenia    [niniejszej    pracy]  podstawowe
     zarówno  autora   [tej  pracy],   jak  i w
 tej pracy,   jak  i w [innych  pracach]   zadobserwowano
```

Figure 6: Example of *con* output

4.4. UTT and standard UNIX text tools

Thanks to the fact that UTT files may be easily processed by standard UNIX text tools, more sophisticated computations may be performed without additional programming. For example, to obtain the frequency list of adjective lexemes preceding a form of the nominal lexeme *praca* (work), the following command may be used:

```
tok  | lem  --one  \
| ser  -m  -e  "tag(<ADJ>)    space   lexeme(praca)"     \
| egrep  ',ADJ/'   \
| sed  -r  's/^. *[:;]([[:alpha:]]+),ADJ\/.       *$/\1/ ' \
| sort  | uniq  -c  | sort  -n  -r
```

ser with `-m` option outputs only matching fragments of the input file. The lines containing adjectives are selected by `egrep`, the base-forms are then extracted by `sed`, sorted, counted, and arranged in desired order.

[3] on a PC with Celeron 1.8 GHz, 256 MB RAM, disk read time 22MB/sec.

[4] *piszemy* ([we] write) is a correct word and present in the dictionary, *gitesowe* (cool) is a slang adjective, absent in the dictionary, *progrumy* (programs) contains a spelling error.

5. Languages and tagsets

The UTT package has originally been developed for Polish. Polish dictionary data for *lem*, *gue*, and *cor* is derived from the large-coverage morphological dictionary Polex/PMDBF (Vetulani, 2000) and is made available for use with the package.

The adaptation of UTT to other languages is possible, provided that dictionaries in appropriate format are supplied (scripts for compiling the dictionaries into the binary format used by component programs make part of the package). Recently UTT has been successfully ported to Portuguese. The dictionary data for *lem*, *gue*, and *cor* components was derived from the UNITEX-PB dictionary (Muniz et al., 1995), which is available under GNU Licence. The format of tags (Intex/Unitex-type) was retained. A single multi-language installation of UTT, equipped in dictionary data for several languages (Polish and Portuguese at the moment) is now being in test phase.

The set of macrodefinitions used by the search tools to expand high-level terms into character-level regular expressions may by freely extended or modified by the user. This feature allows to modify the syntax of search patterns and allows for processing files with different types of annotation.

Adaptation of the search components (*ser* and *grp*) to another tag format is accomplished by redefining the expansion of <**...**> -expressions into character-level regular expressions matching appropriate sets of tags. Technically this problem reduces to implementation of a short script performing the expansion. The only restriction on the tag form, which can be handled by UTT programs is that space, comma and semicolon characters are not allowed in tags. So far, UTT has worked with PMDB-type (eg. `N/NsGpCa`) and Intex/Unitex-type (eg.`N+Pr:ms`) tags.

6. Conclusion

We have presented a language processing toolkit, called UTT. Thanks to the adopted architecture, which permits to connect the components in various ways, UTT may be used to perform numerous different text processing tasks. The system is also easily extensible: new components may be incorporated into the system provided that they respect the i/o file format structure.

The price paid for flexibility and openness of the UTT package is its difficult portability to other, non-UNIX-like, systems. Moreover, users not accustomed to command line work-style may find UTT's interface unfriendly. Development of a graphical user interface is planned in order to make the UTT functionality more accessible to less technically-oriented users. This, however, will take place after the system attains stability.

The first confrontation of the toolkit with serious LR-related research was its use in the *SyntLex* project ((Vetulani et al., 2006)) for retrieving candidates for support verbs for a list of already identified predicative nouns on the basis of IPI PAN Corpus. The feature which proved to be crucial while using UTT to support lexicographical investigations was the possibility to automatically run the search processes from the level of shell scripts (eg. when a series of 50 000 search tasks for different pairs predicative-noun—syntactic-pattern had to be performed).

7. Acknowledgements

The authors wish to thank the following people: Jorge Gilberto, João Gomes and Luis Pedro for their collaboration in porting UTT to Portuguese, Justyna Walkowska for the implementation of the `con` component, and Filip Graliński for his valuable remarks and suggestions.

8. References

Marcelo Muniz, Maria das Graças Volpe Nunes, and Eric Laporte. 1995. Unitex-PB, a set of flexible language resources for brazilian portuguese. In *Proceedings of the III Workshop em Tecnologia da Informação e da Linguagem Humana - TIL, XXV Congresso da SBC*, São Leopoldo.

Tomasz Obrębski. 2006. Searching text corpora with grep. In *(to appear) Intelligent Information Processing and Web Mining Proceedings of the International IIS: IIPWM 06 Conference, Ustronie*, Advances in Soft Computing. Springer-Verlag.

Kemal Oflazer. 1996. Error-tollerant finite state recognition with applications to morphological analysis and spelling correction. *Computational Linguistics*, 22(1):73–89.

Adam Przepiórkowski and Zygmunt Krynicki et al. 2004. Search tool for corpora with positional tagsets and ambiguities. In *The Proceedings of LREC 2004*, pages 1235–1238.

Adam Przepiórkowski. 2004. *The IPI PAN Corpus*. IPI-PAN.

Max Silberztein. 1993. *Dictionnaires électroniques et analyse automatique de textes. Le système INTEX*. MASSON, Paris.

Grażyna Vetulani, Zygmunt Vetulani, and Tomasz Obrębski. 2006. Syntactic lexicon of polish predicative nouns. In *this volume*.

Zygmunt Vetulani. 2000. Electronic language resources for polish: POLEX, CEGLEX and GRAMLEX. In *Gavrilidou et al: Second International Conference on Language Resources and Evaluation, Athens, 30.05-2.06.2000*, pages 367–374, ELRA, Paris.