# Open source morphological analyzer

Németh László*, Halácsy Péter* Kornai András**, Trón Viktor***

**Kivonat** The HunTools natural language processing toolkit supports spellchecking, stemming, morphological analysis and generation in a set of language-neutral routines. The core engine of the LT toolkit is based on the well-known open-source spellchecker OpenOffice.org MySpell, but has several additional functionalities. It supports morphological analysis, also in case of languages with rich morphology and compounding. Both the Hungarian-specific and the language-neutral parts of the system are available under open source LGPL license.

Keywords: **Morphological analysis, open source, Ispell, Myspell, Unicode**

## 1. Introduction

The HunTools natural language processing toolkit emerged from the SzóSzablya morphological analyzer project at the the Budapest Institute of Technology Media Education and Research Center[3,5]. In this paper we concentrate on the architecture of the MorphBase morphological component which supports spellchecking, stemming, morphological analysis, and generation in a set of language-neutral routines. Both the Hungarian-specific and the language-neutral parts of the system are available under the open source LGPL license.

The core engine of MorphBase is a descendant of the well-known International Ispell spellchecker, which identifies correctly spelled words by first stripping affixes according to a rule-system and then looking up the stems from a lexicon. Our code base comes from the open source `OpenOffice.org MySpell`, a portable and thread-safe C++ library reimplementation of `International Ispell`. Both the rule-system and the lexicon are specified as input files and compiled off-line. Our improved version is similarly language independent (and compatible with Myspell file formats) but has significant additional functionality.

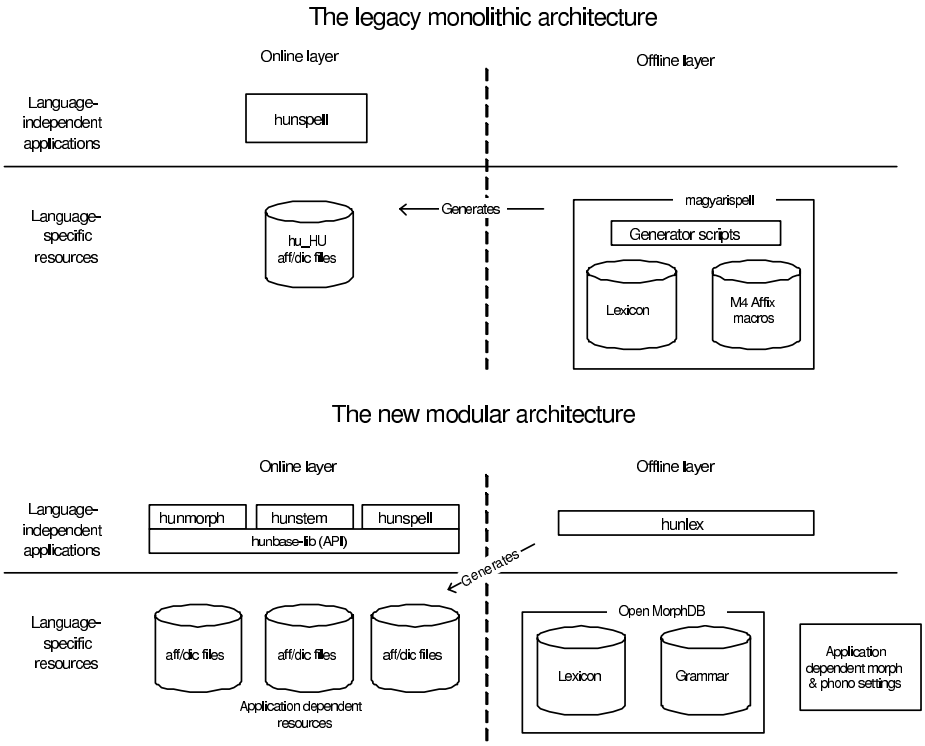
We have also focused on solving several other morphological and orthographical problems in a simple matter, for example Morphbase knows and handles null morphemes, circumfixes, fogemorphemes, homonyms, special compounding rules, etc. and Unicode encoding.

* Budapesti Műszaki Egyetem Média Oktató és Kutató Központ, {nemeth,halacsy}@mokk.bme.hu
** MetaCarta Inc., andras@kornai.com
*** International Graduate College, Saarland University and University of Edinburgh, v.tron@ed.ac.uk

## 2. Architecture of Huntools

### 2.1. Language-independent applications

Though the names `HunSpell` spell checker, `HunStem` stemmer, `HunMorph` morphological analyser suggest Hungarian orientation, in the spirit of `Ispell`, our project keeps technology and lexical resources perfectly separated (see figure).




1. ábra. Architecture of Huntools

As a result, all the modules of our LT toolkit are directly applicable to other languages provided that lexical resources are available. Since Myspell resources are available for more than 50 languages, our stemmer and spellchecker can be readily used for other languages.

### 2.2. HunLex

The sheer size and redundancy of precompiled morphologies make modifications very difficult and debugging nearly impossible. Maintaining these resources without a principled framework for off-line resource compilation is hopeless, witness

`Magyar Ispell`, the Hungarian `MySpell` resource which resorts to a clever (from a maintainability perspective, way too clever) mix of shell scripts, M4 macros, and hand-written pieces of `MySpell` resources.

To remedy this problem we devised an offline resource compilation tool, `HunLex`, which given a central lexical database and a morphological grammar can create resources for the applications according to a wide range of configurable parameters. `HunLex` is a language-independent pre-processing framework for a rule-based description of morphological grammars ([6]).

## 3. Introduction to Morphbase

Showing examples is the best method to introduce capabilities of Morphbase. We will describe shortly the format of the resource (affix and dictionary file), give and explain examples about several new features of our Ispell's descendant.

### 3.1. Basic format of Morphbase resource

Morphbase requires two files to define the language that it is analysing. The first file is a dictionary containing words for the language, and the second is an "affix" file that defines the meaning of special flags in the dictionary.

A dictionary file (*.dic) contains a list of words, one per line. First line of the dictionaries (except personal dictionaries) contains the word count. Each word may optionally be followed by a slash ("/") and one or more flags, which represents affixes or special attributes. Default flag format is a single (usually alphabetic) character. In Hunspell dictionary file, there is also an optional morphological field separated by tabulator. Morphological desciptions have custom format.

An affix file (*.aff) may contain a lot of optional attributes. For example, *SET* is used for setting the character encodings of affix and dictionary files. *TRY* sets the change characters for suggestions. *REP* sets a replacement table for multiple character corrections in suggestion mode. *PFX* and *SFX* definies prefix and suffix classes named with affix flags.

The following affix file example definies UTF-8 character encoding. *TRY* suggestions differ from the bad word with an English letter or an apostrophe. With these REP definitions, Hunspell can suggest the right word form, when the misspelled word contains f instead of ph and vice versa.

```
SET UTF-8
TRY esianrtolcdugmphbyfvkwzESIANRTOLCDUGMPHBYFVKWZ'

REP 2
REP f ph
REP ph f

PFX A Y 1
```

```
PFX A 0 re .

SFX B Y 2
SFX B 0 ed [^y]
SFX B y ied y
```

There are two affix classes in the dictionary. Class A definies an 're-' prefix. Class B definies two -ed' suffixes. First suffix can be added to a word if the last character of the word isn't 'y'. Second suffix can be added to words terminated with an 'y'. (See details later.) The following dictionary file uses these affix classes.

```
3
hello
try/B
move/AB
```

All accepted words with this example: *hello*, *try*, *tried*, *move*, *moved*, *remove*, *removed*.

### 3.2. Options in affix file

Nearly every single option in the affix file corresponds a Morphbase feature. We can give a shortly abstract about these features by introducing options.

**SET encoding** Set character encoding of words and morphemes in affix and dictionary files. Possible values: UTF-8, ISO8859-1 – ISO8859-10, ISO8859-14, KOI8-R, KOI8-U, microsoft-cp1251, ISCII-DEVANAGARI.

**TRY characters** Hunspell can suggest right word forms, when those differs from the bad form by one TRY character. Parameter of TRY is case sensitive.

**LANG langcode** Set language code. In Hunspell may be language specific codes enabled by LANG code. At present there are az_AZ, de_DE, hu_HU, TR_tr specific codes in Hunspell (see program source).

**FLAG value** Set flag type. Default value is character. The *long* value sets the 2-character flag type, the *num* sets the decimal number flag type. Decimal flags are numbered from 1 to 65535, and separated by comma.

**COMPOUNDMIN num** Minimum length of words in compound words. Default value is 3 letter.

**COMPOUNDFLAG flag** Words signed with COMPOUNDFLAG may be in compound words (except when word shorter than COMPOUNDMIN). Affixes with COMPOUNDFLAG also permits compounding of affixed words.

**COMPOUNDBEGIN flag** Words signed with COMPOUNDFLAG (or with a signed affix) may be first elements in compound words.

**COMPOUNDLAST flag** Words signed with COMPOUNDFLAG (or with a signed affix) may be last elements in compound words.

**COMPOUNDMIDDLE flag** Words signed with COMPOUNDFLAG (or with a signed affix) may be middle elements in compound words.

**ONLYINCOMPOUND flag** Suffixes signed with ONLYINCOMPOUND flag may be only in inside of compounds (Fuge-elements in German, fogemorphemes in Swedish).

**CIRCUMFIX flag** Affixes signed with CIRCUMFIX flag may be on a word when this word also has a prefix with CIRCUMFIX flag and vice versa.

**COMPOUNDFORBIDFLAG flag** Suffixes with this flag forbid compounding of the affixed word.

**COMPOUNDROOT flag** COMPOUNDROOT flag signs the compounds in the dictionary (Now it is used only in the Hungarian language specific code).

**COMPOUNDWORDMAX number** Set maximum word count in a compound word. (Default is unlimited.)

**FORBIDDENWORD flag** This flag signs forbidden word form. Because affixed forms are also forbidden, we can substract a subset from set of the accepted affixed and compound words.

**PSEUDOROOT flag** This flag signs virtual stems in dictionary. Only affixed forms of these words will be accepted by Hunspell. Except, if dictionary word has a homonym or a zero affix (null morpheme).

**WORDCHARS characters** WORDCHARS extends tokenizer of Hunspell command line interface with additional word character. For example, dot, dash, n-dash, numbers, percent sign are word character in Hungarian.

**LEMMA_PRESENT flag** Generally, there are dictionary words as lemmas in output of morphological analysis. Sometimes dictionary words are not lemmas, but affixed (not real) stems and virtual stems. In this case lemmas (real stems) need to put into morphological description, and forbid not real lemmas in morphological analysis adding LEMMA_PRESENT flag to dictionary words.

**COMPOUNDSYLLABLE max_syllable vowels** Need for special compounding rules in Hungarian. First parameter is the maximum syllable number, that may be in a compound, if words in compounds are more than COMPOUNDWORDMAX. Second parameter is the list of vowels (for calculating syllables).

**SYLLABLENUM flags** Need for special compounding rules in Hungarian.

**REP number_of_replacement_definitions**

**REP what replacement** We can define language-dependent phonetic information in the affix file (.aff) by a replacement table. First REP is the header of this table and one or more REP data line are following it. With this table, MySpell can suggest the right forms for the typical faults of spelling when the incorrect form differs by more, than 1 letter from the right form. For example a possible English replacement table definition to handle misspelt consonants:

```
REP 8
REP f ph
REP ph f
REP f gh
REP gh f
```

```
REP j dg
REP dg j
REP k ch
REP ch k
```

Replacement table is also usable in robust morphological analysis (accepting bad forms) or stricter compound word support in spell checking (forbidding generated compound words, if they are simple words with typical fault at the same time).

**PFX flag cross_product number**
**PFX flag stripping prefix condition morphological_description**
**SFX flag cross_product number**
**SFX flag stripping suffix condition morphological_description** An affix is either a prefix or a suffix attached to root words to make other words. We can define affix classes with arbitrary number affix rules. Affix classes signed with affix flags. First line of an affix class definition is the header. Fields of an affix class header:

1. Option name (PFX or SFX).
2. Flag (name of the affix class).
3. Cross product (permission to combinate prefixes and suffixes). Possible values: Y (yes) or N (no).
4. Line count of the following rules.

Fields of the affix rules:

1. Option name.
2. Flag.
3. Stripping characters from beginning (at prefix rules) or end (at suffix rules) of the word.
4. Affix (optionally with flags of continuation classes, separated by a slash).
5. Condition.
6. Custom morphological description.

Zero stripping or affix are indicated by zero. Zero condition is indicated by dot. Condition is a simplified, regular expression-like pattern, which must be met before the affix can be applied. (Dot signs arbitrary character. Characters in braces sign an arbitrary character from the character subset. Dash hasn't got special meaning, but circumflex (^) next the first brace sets complementer character set.)

## 4. Morphological analysis

Hunmorph's affix rules has got an optional morphological description field. There is a similar optional field in dictionary file, separated by tabulator:

```
word/flags     morphology
```

We defines a simple resource with morphological informations.
Affix file:

```
SFX X Y 1
SFX X 0 able . +ABLE
```

Dictionary file:

```
drink/X   [VERB]
```

Test file:

```
drink
drinkable
```

Test:

```
$ hunmorph test.aff test.dic test.txt
drink:     drink[VERB]
drinkable: drink[VERB]+ABLE
```

You can see in the example, that the analyzer concatenates the morphological fields in *item and arrangement* style.

## 5. Twofold suffix stripping

Ispell's original algorithm strips only one suffix. Hunmorph can strip another one yet.

The twofold suffix stripping is a significant improvement in handling of immense number of suffixes, that characterized the agglutinative languages.

Extending the previous example by adding a second suffix (affix class Y will be the continuation class of able suffix):

```
SFX Y Y 1
SFX Y 0 s . +PLUR

SFX X Y 1
SFX X 0 able/Y . +ABLE
```

Dictionary file:

```
drink/X   [VERB]
```

Test file:

```
drink
drinkable
drinkables
```

Test:

```
$ hunmorph test.aff test.dic test.txt
drink:      drink[VERB]
drinkable:  drink[VERB]+ABLE
drinkables: drink[VERB]+ABLE+PLUR
```

Theoretically with the twofold suffix stripping needs only the square root of the number of suffix rules, compared with a MySpell implementation. In our practice, we could have elaborated the Hungarian inflectional morphology with twofold suffix stripping. (Note: In Hunlex preprocessor's grammar can be use not only twofold, but multiple suffix slitting.)

## 6. Extended affix classes

Hunmorph can handle more than 65000 affix classes. There are two new syntax for giving flags in affix and dictionary files.

*FLAG long* command sets 2-character flags:

```
FLAG long
SFX Y1 Y 1
SFX Y1 0 s 1
```

Dictionary record with the Y1, Z3, F? flags:

```
foo/Y1Z3F?
```

*FLAG num* command sets numerical flags separated by comma:

```
FLAG num
SFX 65000 Y 1
SFX 65000 0 s 1
```

Dictionary example:

```
foo/65000,12,2756
```

## 7. Homonyms

Hunmorph's dictionary can contain repeating elements that is homonyms:

```
work/A    [VERB]
work/B    [NOUN]
```

An affix file:

```
SFX A Y 1
SFX A 0 s . +SG3

SFX B Y 1
SFX B 0 s . +PLUR
```

Test file:

```
works
```

Test:

```
> works
work[VERB]+SG3
work[NOUN]+PLUR
```

This feature also gives a way to forbid illegal prefix/suffix combinations in difficult cases.

## 8. Prefix–suffix dependencies

An interesting side-effect of multi-step stripping is that the appropriate treatment of circumfixes now comes for free. For instance, in Hungarian, superlatives are formed by simultaneous prefixation of *leg-* and suffixation of *-bb* to the adjective base. A problem with the one-level architecture is that there is no way to render lexical licensing of particular prefixes and suffixes interdependent, and therefore incorrect forms are recognized as valid, i.e. **legvén* = *leg* + *vén* 'old'. Until the introduction of clusters a special treatment of the superlative had to be hardwired in the earlier `HunSpell` code. This may have been legitimate for a single case, but in fact prefix–suffix dependences are ubiquitous in category-changing derivational patterns (cf. English *payable*, *non-payable* but **non-pay* or *drinkable*, *undrinkable* but **undrink*). In simple words, here, the prefix *un-* is legitimate only if the base *drink* is suffixed with *-able*. If both these patters are handled by on-line affix rules and affix rules are checked against the base only, there is no way to express this dependency and the system will necessarily over- or undergenerate.

In next example, suffix class R have got a prefix continuation' class (class P).

```
PFX P Y 1
PFX P   0 un . [prefix_un]+

SFX S Y 1
SFX S   0 s . +PL

SFX Q Y 1
SFX Q   0 s . +3SGV

SFX R Y 1
SFX R   0 able/PS . +DER_V_ADJ_ABLE
```

Dictionary:

```
2
drink/RQ [verb]
drink/S [noun]
```

Morphological analysis:

```
> drink
drink[verb]
drink[noun]
> drinks
drink[verb]+3SGV
drink[noun]+PL
> drinkable
drink[verb]+DER_V_ADJ_ABLE
> drinkables
drink[verb]+DER_V_ADJ_ABLE+PL
> undrinkable
[prefix_un]+drink[verb]+DER_V_ADJ_ABLE
> undrinkables
[prefix_un]+drink[verb]+DER_V_ADJ_ABLE+PL
> undrink
Unknown word.
> undrinks
Unknown word.
```

## 9. Circumfix

Conditional affixes implemented by continuation class are not enough for circumfixes, because a circumfix is one affix in morphology. We also need CIRCUMFIX option for correct morphological analysis.

```
# circumfixes: ~ obligate prefix/suffix combinations
# superlative in Hungarian: leg- (prefix) AND -bb (suffix)
# nagy, nagyobb, legnagyobb, legeslegnagyobb
# (great, greater, greatest, most greatest)

CIRCUMFIX X

PFX A Y 1
PFX A 0 leg/X .

PFX B Y 1
PFX B 0 legesleg/X .

SFX C Y 3
SFX C 0 obb . +COMPARATIVE
```

```
SFX C 0 obb/AX . +SUPERLATIVE
SFX C 0 obb/BX . +SUPERSUPERLATIVE
```

Dictionary:

```
1
nagy/C [MN]
```

Analysis:

```
> nagy
nagy[MN]
> nagyobb
nagy[MN]+COMPARATIVE
> legnagyobb
nagy[MN]+SUPERLATIVE
> legeslegnagyobb
nagy[MN]+SUPERSUPERLATIVE
```

## 10. Compounds

Allowing free compounding yields decrease in precision of recognition, not to mention stemming and morphological analysis. Although lexical switches are introduced to license compounding of bases by `Ispell`, this proves not to be restrictive enough. For example:

```
# affix file
COMPOUNDFLAG X

2
foo/X
bar/X
```

With this resource, *foobar* and *barfoo* also are accepted words.

This has been improved upon with the introduction of direction-sensitive compounding, i.e., lexical features can specify separately whether a base can occur as leftmost or rightmost constituent in compounds. This, however, is still insufficient to handle the intricate patterns of compounding, not to mention idiosyncratic (and language specific) norms of hyphenation.

The `MySpell` algorithm currently allows any affixed form of words which are lexically marked as potential members of compounds. `Hunspell` improved upon this, and its recursive compound checking rules makes it possible to implement the intricate spelling conventions of Hungarian compounds. For example, using COMPOUNDWORDMAX, COMPOUNDSYLLABLE, COMPOUNDROOT, SYLLABLENUM options can be set the noteworthy Hungarian '6–3' rule. Further example in Hungarian, derivate suffixes often modify compounding properties, hence in Hunmorph can be also use the compounding flags on

affixes, and there is also a special flag (COMPOUNDFORBIDFLAG) to prohibit compounding of the derivations.

We also need several Hunmorph features for handling German compounding:

```
# German affix file

# set language for handling compound words with dash
LANG de_DE

COMPOUNDBEGIN U
COMPOUNDMIDDLE V
COMPOUNDEND W

# for German fogemorphemes (Fuge-element)
ONLYINCOMPOUND X

# for decapitalizing nouns with fogemorphemes
CIRCUMFIX Y

# for handling Fuge-elements with dashes (Arbeits-)
# dash will be a special word
COMPOUNDMIN 1
WORDCHARS -

# Fuge-element, first position
# (without decapitalize) <Arbeitscomputer>
SFX I Y 1
SFX I 0 s/UX .

# Fuge-element, middle position
# (with decapitalize). <Computerarbeitsplatz>
SFX J Y 1
SFX J 0 s/VYBX .

# for forbid exceptions <*Arbeitsnehmer>
FORBIDDENWORD Z

# decapitalizing prefix, in middle of compounds
PFX A Y 29
PFX A A a/VX A
...
PFX A Z z/VX Z

# decapitalizing 'circumfix', with Fuge-element
PFX B Y 29
PFX B A a/VXY A
```

```
...
PFX B Z z/VXY Z

# decapitalizing prefix, in end of compounds
PFX C Y 29
PFX C A a/WX A
...
PFX C Z z/WX Z
```

Example dictionary:

```
4
Arbeit/IJKC
Computer/UAC
-/W
Arbeitsnehmer/Z
```

Accepted compound compound words with the previous resource:

```
Computer
Arbeit
Arbeits-
Computerarbeit
Computerarbeits-
Arbeitscomputer
Computerarbeitscomputer
Arbeitscomputerarbeit
Computerarbeits-Computer
```

Not accepted compoundings:

```
computer
arbeit
Arbeits
arbeits
ComputerArbeit
ComputerArbeits
Arbeitcomputer
ArbeitsComputer
Computerarbeitcomputer
ComputerArbeitcomputer
ComputerArbeitscomputer
Arbeitscomputerarbeits
Computerarbeits-computer
Arbeitsnehmer
```

This solution is still not ideal, however, and will be replaced by a pattern-based compound-checking algorithm which is closely integrated with input buffer

tokenization. Patterns describing compounds come as a separate input resource that can refer to high-level properties of constituent parts (e.g. the number of syllables, affix flags, and containment of hyphens). The patterns are matched against potential segmentations of compounds to assess wellformedness.

## 11. Character encoding

### 11.1. Problems with the 8-bit encoding

Both `Ispell` and `Myspell` use 8-bit ASCII character encoding, which is a major deficiency when it comes to scalability. Although a language like Hungarian has a standard ASCII character set (ISO 8859-2), it fails to allow a full implementation of Hungarian orthographic conventions. For instance, the '–' symbol (n-dash) is missing from this character set contrary to the fact that it is not only the official symbol to delimit paranthetic clauses in the language, but it can be in compound words as a special 'big' hyphen.

MySpell has got some 8-bit encoding tables, but there are languages without standard 8-bit encoding, too. For example, a lot of African languages have non-latin or extended latin characters.

Similarly, using the original spelling of certain foreign names like *Ångström* or *Molière* is encouraged by the Hungarian spelling norm, and, since characters 'Å' and 'è' are not part of ISO 8859-2, when they combine with inflections containing characters only in ISO 8859-2 (like elative *-ből*, allative *-től* or delative *-ről*), these result in words (like *Ångströmről* or *Molière-ről*.) that can not be encoded using any single ASCII encoding scheme.

The problems raised in relation to 8-bit ASCII encoding have long been recognized by proponents of Unicode. Unfortunately, switching to Unicode (e.g., UTF-16 encoding) would require a great deal of code optimization and would have an impact on the efficiency of the algorithm. The Dömölki algorithm [1] used in checking affixation conditions utilizes 256-byte character arrays, which would grow to 64k with Unicode encoding. Since online affixation for a richly agglutinative language can easily have several hundred[1], such arrays, switching to Unicode would incur high resource costs. Nonetheless, it is clear that trading efficiency for encoding-independence has its advantages when it comes a truely multi-lingual application, therefore it was among our plans for a long while to extend the architecture in this direction.

### 11.2. A hybrid solution

Recently we implemented successfully a memory and time efficient Unicode handling, with hybrid string manipulation and condition checking.

[1] In the case of the standard Hungarian resources we use, this number is ca. 300 or more since redundant storage of structurally identical affix patterns improves efficiency.

Affixes and words are stored in UTF-8, during the analysis are handled in mostly UTF-8, in condition checking and suggestion are converted to UTF-16.

Dömölki-algorithm is used for storing and checking 7-bit ASCII (ISO 646) condition character, and sorted UTF-16 lists for other Unicode character of condition patterns.

## Conclusion

The core engine of the Huntools toolkit is descendant of the widely adopted International Ispell and OpenOffice.org Myspell, but has several additional functionalities.

In a simple manner, we enabled the output of stripped forms, thereby creating a stemmer. Next, we enabled alternative analyses, both in stemming and providing a full morphological analysis. Furthermore, we replaced the simple the simple one-pass affix stripping mechanism of ispell by a recursive system in which affixes can be stripped in as many layers as needed. This results in considerable simplification of the lexical resource, as well as increased linguistic transparency and maintainability. Finally, we implemented new features for handling special morphological and ortographical problems of different languages, for example circumfixes and 6–3 compounding rule in Hungarian and fogemorphemes in German languages.

In the paper we try to introduce the capability of our morphological analyzer by several examples. We also try to assess the merits and limitations of our tools with special attention to the possibility of applying them to bootstrapping resources in other languages, especially ones with rich morphology, difficult orthographical rules in compounding or special alphabets.

## Acknowledgements

Our project is funded by an ITEM grant from the Hungarian Ministry of Informatics and Telecommunications, and benefits greatly from logistic and infrastructural support of MATÁV Rt. and Axelero Internet.

## Hivatkozások

1. B Dömölki. Algorithms for the recognition of properties of sequences of symbols. *USSR Computational & Mathematical Physics*, 5(1):101–130, 1967. Pergamon Press, Oxford.
2. Péter Halácsy, András Kornai, László Németh, András Rung, István Szakadát, and Viktor Trón. Szógyakoriság és helyesírás-ellenőrzés [word frequency and spellchecker accuracy]. In *Proceedings of the 1st Hungarian Computational Linguistics Conference*, pages 211–217. Szegedi Tudományegyetem, 2003.
3. Péter Halácsy, András Kornai, László Németh, András Rung, István Szakadát, and Viktor Trón. Creating open language resources for Hungarian. In *Proceedings of Language Resources and Evaluation Conference (LREC04)*. European Language Resources Association, 2004.

4. Németh László. Magyar Ispell – Válasz a Helyes-e?-re. In *IV. GNU/Linux szakmai konferencia*, pages 99–107. Linux-felhasználók Magyarországi Egyesülete, 2002.
5. Németh László. A Szószablya fejlesztés. In *V. GNU/Linux szakmai konferencia*. Linux-felhasználók Magyarországi Egyesülete, 2003.
6. Trón Viktor. Hunlex - morfológiai szótárkezelő rendszer. In *II Magyar Számítógépes Nyelvészeti Konferencia*, 2004.