Leveraging the open source ispell codebase for minority language analysis

László Németh*, **Viktor Trón**[†], **Péter Halácsy***, **András Kornai**[‡], **András Rung** *, **István Szakadát***

*Budapest Institute of Technology Media Research and Education Center
{nemeth,halacsy,rung,szakadat}@mokk.bme.hu
[†]International Graduate College, Saarland University and University of Edinburgh, v.tron@ed.ac.uk
[‡]MetaCarta Inc., andras@kornai.com

Abstract

The *ispell* family of spellcheckers is perhaps the single most widely ported and deployed open-source language tool. Here we describe how the *SzóSzablya* 'WordSword' project leverages `ispell`'s Hungarian descendant, `HunSpell`, to create a whole set of related tools that tackle a wide range of low-level NLP-related tasks such as character set normalization, language detection, spellchecking, stemming, and morphological analysis.

1. Introduction

Over the years, open source unix distributions have become the definitive repositories of tried and tested algorithms. In the area of natural language processing, wellformedness of words is typically checked by the `ispell` family of spellcheckers that goes back to Gorin's `spell` program (see Peterson 1980), a spellchecker for English written in PDP-10 assembly. Since at the core of spellchecking is a method for accurate word recognition, it is an ideal platform both for reaching "down" toward language identification and for reaching "up" toward stemming and morphological analysis. The *SzóSzablya* 'WordSword' project at the Budapest Institute of Technology leverages the `ispell` methods with the goal to extend them to a general toolkit applicable to various low-level NLP-related problems other than spell-checking such as language detection, character set normalization, stemming, and morphological analysis.[1]

The algorithms described here go back to the roots of the `spell -- ispell -- International Ispell -- MySpell -- HunSpell` development. The linguistic theory implicit in much of the work has an even deeper historical lineage, going back at least to the Bloomfield–Bloch–Harris development of structuralist morphology via Antal's (1961) work on Hungarian. Despite our indebtedness to these traditions, this paper does not attempt to faithfully trace the twists and turns of the actual history of ideas, rather it offers only a rational reconstruction of the underlying logic.

A high performance spellchecker can easily be leveraged for language identification, and we have relied heavily on `HunSpell` both for this purpose and for overall quality improvement in creating a gigaword Hungarian corpus (see the main conference paper paper Halácsy et al 2004). Orthographic form and, by implication, spellchecking technology, remains the Archimedean point of natural language text processing both "downward" and "upward". Here we will concentrate entirely on the "upward" developments leading to `HunStem`, a full featured industrial strength stemmer that supports large-scale Information Retrieval applications, and eventually to `HunMorph`, an open source morphological analyzer.[2] Though the names `HunSpell` and `HunStem` suggest Hungarian orientation, in the spirit of `ispell` our project keeps the technology perfectly separated from lexical resources, making the tools are directly applicable to other languages provided that lexical databases are available. Resources for the applications can be compiled from a single lexical database and morphological grammar with the help of the `HunLex` resource compilation tool.

The paper is structured as follows. Section 2 provides a brief introduction to the morphological analysis/generation problem from the perspective of spellchecking, and discusses how the affix-flag mechanism introduced to `ispell` by Ackerman in 1978[3] has been modified to deal with multi-step affix stripping to attack the problem of languages with rich morphology. Section 3 describes how, by enabling multiple analyses, treatment of homonyms, and flexible output of stem information, the general framework of `HunSpell` has been extended to support stemming. In the concluding Section 4 we describe how the codebase can be leveraged even further, to support detailed morphological analysis.

2. The morphological OOV problem

The simplest spellchecker, both conceptually and in terms of optimal runtime performance, is a list of all correctly spelled words. Acceleration and error correction techniques based on hashes, tries, and finite automata have been extensively studied, and the implementor can choose from a variety of open source versions of these techniques. Therefore the spellchecking problem could be reduced, at least conceptually, to the problem of listing the correct words, whereby errors of the spellchecker are reduced to out of vocabulary (OOV) errors. A certain amount of OOV error is inevitable: new words are coined all the time, and the supply of exotic technical terms and proper names is inexhaustible. But as a practical matter, developers encounter

[1] Aversano et al 2002 is the only related attempt we know of.

[2] For further upward developments such as named entity extraction, parsing, or semantic analysis, orthography gradually loses its grip over the problem domain, but none of these higher-level developments are feasible without tackling the low-level issues first.

[3] For the history of `ispell/MySpell`, see the man pages.

OOV errors early on from another source: morphologically complex words such as compounds and affixed forms.

The ability to reverse compounding and affixation has a very direct payoff in terms of reducing memory footprint, and it is no surprise that affix stripping ability was built into `ispell` early on. Initially, `(i)spell` only used heuristics for affix stripping before looking up hypothesized stems in a base dictionary. This was substantially improved by the introduction of *switches* (in linguistic terms these would be called *privative lexical features*) that license particular affix rules and thus help eliminate spurious hits resulting from the unreliable heuristic method.

In 1988 Geoff Kuenning extended affix flags to license sets of affix rules. In this table-driven approach, affix flags are interpreted as lexical features that indicate morphological subparadigm membership. This method of affix compression allowed for less redundant storage and efficient run-time checking of a great number of affixes, thereby enabling `ispell` to tackle languages with more complex morphological systems than English. After major modifications of the code, the first multi-lingual version of `ispell` was released in 1988.

`Ispell` can also handle compounds and there is even the possibility of specifying lexical restrictions on compounding, also implemented as switches in the base dictionary. For some languages, a rich set of compound constructions allow for productive extensions of the base vocabulary, and this feature is indispensable in mitigating the OOV problem. Language-specific word-lists and affix rules for `ispell`, with added switch information as necessary, have been compiled for over 50 languages so far. Our development started with providing open source spell-checking for Hungarian. Our spellchecker, `HunSpell` is based on `MySpell`, a portable and thread-safe C++ library reimplementation of `ispell` written by Kevin Hendricks. We chose `MySpell` as the core engine for our development both because of its implementational virtues, and because the non-restrictive BSD license significantly enhances its potential in open source development and large-scale code reuse.[4]

The lexical resources of `MySpell` (the affix file and the dictionary file) are processed at runtime, which makes them directly portable across various platforms. A line in the affix file represents an affix rule from a generation point of view. It specifies a regexp-like pattern which is matched against the beginning or end of the base for prefix and suffix respectively, a string to strip from, and the actual affix string to append to the input base. A special indexing technique, the Dömölki algorithm is used in checking affixation conditions (Dömölki 1967) to pick applicable affix rules in parsing. A pseudo-stem is hypothesized by reverse application of the affix rule (i.e., stripping the append string and appending the strip string) which is looked up in the dictionary. A line in the dictionary file represents a lexical entry, i.e., a base form associated with a set of affix flags. If the hypothesized base is found in the dictionary after the application of an affix rule, in the last step it is checked whether its flags contain the one the affix rule is assigned to.

Though the ispell algorithm performs affix stripping and lexical lookup very efficiently, the implementation does not scale well to languages with rich morphology. `ispell` lexical resources actually exist for some languages with famously rich productive morphology such as Estonian, Finnish, and Hungarian, but it is suggestive that the latter two languages use enhancements over `MySpell` in their native OpenOffice.org releases for spellchecking.[5] The Hungarian version uses our development, the `HunSpell` library which incorporates various spell-checking features specifically needed to correctly handle Hungarian orthography – we turn to these now.

So far we spoke of affixes only in the sense of edge-aligned substrings (prefixes and suffixes), but in languages with complicated combinatorial morphology affix rules might stand for intricate clusters of affix morphemes (the sense of affix used in linguistics). Such morphotactic complexity, a hallmark of rich productive morphology, often makes it difficult to list all legitimate affix combinations, let alone produce them automatically: 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 virtually a hopeless enterprise, witness `magyarispell`, the Hungarian `MySpell` resource[6] 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 off-line resource compilation tool 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 morphology (details about grammar specifications and configuration options of `HunLex` would go beyond this paper).

To handle all the productive inflections `magyarispell` requires about 20 thousand combined entries. Extending this database to incorporate productive derivational morphology would mean an order of magnitude increase, as full derivation and inflection can yield ca. 10^3-10^6 word forms from a single nominal base. Taking orthogonal prefix combinations into account would result in another order of magnitude increase, leading to file sizes unacceptable in a practical system.

Using the `magyarispell` resources, on the 5 million word types of the SzóSzablya web corpus (Halácsy et al. 2004), `HunSpell`'s recognition performance is about 96% (OOV is 4%). Taking word frequencies into account OOV is only 0.2% (i.e. recall is near perfect, 99.8%). While these figures are quite reassuring for the central use case of flagging spelling errors, in order to offer high quality replacements we can't ignore rare but perfectly well-formed complex forms. Decreased OOV is also indispensable for wide-coverage morphological analysis and Information Re-

[4] MySpell has been incorporated into OpenOffice.org's office suite, where it replaces the third-party libraries licensed earlier.

[5] The Finnish version is a closed-source licensed binary (see `http://www.hut.fi/ pry/soikko/openoffice/`).

[6] `http://magyarispell.sourceforge.net/`

trieval applications.

To solve the morphological OOV problem HunSpell now incorporates a multi-step sequential affix-stripping algorithm. After stripping an affix-cluster in step i, the resulting pseudo-stem can be stripped of affix-clusters in step $i + 1$. Legitimate strippings can be checked in exactly the same way as for valid online base+affix combinations, and are encoded with the help of switches in the resource file. Implementing this only required a minor extension of the data structure coding affix entries and a recursive call for stripping. Currently this scheme is implemented for two steps (plus lexical lookup) for suffixation plus one for prefixation, but can easily be extended to a fully recursive method.[7] From the structuralist perspective, the clustering step implements a kind of position class analysis (Nida 1949, Harris 1951), and from a generative perspective it implements a simplified version of lexical phonology and morphology (Kiparsky 1982). Besides the well-known theoretical justifications for this style of analysis, there is a compelling practical justification in that the size of the affix table shrinks substantially: with our particular setting for Hungarian, the multi-step resource is the square root of the single-step one in size. HunLex can be configured to cluster any or no set of affixes together on various levels, and therefore resources can be optimized on either speed (toward one-level) or memory use (affix-by-affix stripping).

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.

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. 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. 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.

3. Stemming and morphological analysis

So far, we only touched upon general issues pertaining to the recognition of morphologically complex forms in highly inflecting languages. It is easy to realize, however, that the same general architecture can easily be extended to more sophisticated analysis tools for morphological processing. A straightforward extension we implemented allowed HunSpell to output lexical stems, thereby turning it into a simplistic stemmer.

Practically, stemmers are used as a recall enhancing device for Information Retrieval systems (Kraaij and Pohlmann 1996, Hull 1996). Stemmers ideally conflate semantically related wordforms, so indexing words by their stems effectively expands the relevant search space. The relevance of this ubiquitous NLP technique is greater for languages with rich (inflectional) morphology and/or relatively smaller corpus. Stemmers based on various approximate heuristics (Porter 1980, Paice 1994) are already quite robust and ones based on corpus statistics can be totally language independent (Xu and Croft 1998). However, these methods very often produce nonwords the human interpretation of which is difficult, which makes debugging of false conflations hard. Therefore, once linguistic resources are available, stemming based on linguistically motivated morphological analysis is beneficial at least from a maintainability perspective.

To turn HunSpell into HunStem, a fully functional grammar-based stemmer, we had to address several issues beyond the trivial provision for stem output. First, for the recognition problem relevant in word-based spellchecking, no multiple analyses are needed, so the processing of a word can terminate with the first successful analysis. For any stemmer of practical use, this is insufficient, and coming up with alternative stems for morphologically ambiguous forms is a definitive requirement. This has been implemented and HunStem now performs exhaustive search for analyses and outputs all potential stems.

Second, for stemming it is desirable that homonymous stems be disambiguated if affixation provides the necessary cues. This is usually the case with ambiguous stems belonging to different paradigms or categories like Hun-

[7] For the fully recursive version, in order to guarantee termination, one has to either impose a limit on the number of steps or make sure that the lengths of pseudo-stems that are the result of successive legitimate stripping operations converge to zero.

garian *hal*, which is ambiguous between the verbal reading 'die' and the nominal reading 'fish'. In the original design, string-identical bases are conflated and there is no way to tell them apart once their switch-set licensing various affixes are merged. Fixing this only required minor technical modifications in the code and `HunStem` is now able to handle homonyms and output the correct stem if the base occurs in disambiguating affix pattern. Note that base-licensing of incompatible affixes for homonymous stems is a problem for recognition as well. For instance, in Hungarian, *hal*, used as a verb, can take various verbal affixes, but these cannot cooccur with nominal affixes. The problem is that the architecture is unable to rule out homonymous bases with illegitimate simultaneous cross-category combinations of prefix and suffix such as **meghalam* = *meg* verbal prefix + *hal* 'die V' + *-am* 'POSS_1SG nominal suffix'. Before the correct treatment of homonymous bases was introduced, the only available solution was to list precompiled verbal and nominal paradigms separately for these, which is not only wasteful and error-prone, but also puts productively derived forms out of scope.

Third, and most importantly, the literal output of lexical stems looked up in the dictionary resource after affix stripping may not be optimal for stemming purposes without explicitly addressing the issue of when to terminate the analysis. From the perspective of spellchecking there was no reason to get rid of exactly the affixes one is likely to want to strip in a typical stemming task. Since the division of labor between online (runtime) and offline (compile time) affixation is irrelevant for the recognition task, choices are mainly biased to optimize efficient storage and/or processing rather than to reflect some meaningful coherence of dictionary bases. That is, several morphologically complex forms may be appear as bases (either listed or off-line precompiled) in the original `HunSpell` resource dictionary, which was not optimized for stemming output. For instance, Hungarian exceptional singular accusative *farkat* is linguistically derived from stem *farok* but the ispell analysis was based on a dictionary entry for the plural nominative *farkak* for reasons of efficient coding. In the current system the adjustment of conflation classes, i.e., which words are kept unanalyzed and which affixes are stripped, is therefore flexibly configurable: the precompiler `HunLex`, which replaces our earlier set of `m4` macros, creates lexical resources for the stemmer based on various parameters, which opens the door to the creation of task-dependent stemmers optimized differently for different IR applications.

4. Conclusion

If we aspire to scale open source language technology to a wide range of languages, the problems exemplified by Hungarian are but instances of the general problems one will necessarily encounter along the way, because a substantial proportion of the world's languages (e.g., Altaic, Uralic and Native American languages) are heavily agglutinative. Since scaling to other languages is an important motivation behind developing our toolkit, we believe that even those languages with rich morphology, like Turkish or Basque, which as yet lack MySpell lexical resources, will eventually benefit from our efforts.

The next logical step is a full-fledged morphological analysis tool for Hungarian. Many of the prerequisites of morphological analysis, in particular the flexibility to define the set of morphemes left unanalyzed at compile time, were fulfilled in the course of the `HunStem` development, and a pilot version of `HunMorph` is already operational. In principle, `HunLex` method of dictionary resource precompilation is applicable even to Kimmo-style systems, where the inner loop is based on finite state transduction rather than the generic string manipulation techniques used in `ispell`, but in the absence of a non-restrictive license open source two-level compiler we are not in a position to pursue this line of research.

Acknowledgements

The SzóSzablya 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.

5. References

L. Antal, 1961. A magyar esetrendszer [The Hungarian case system]. *Nyelvtudományi Értekezések*, 29:57–77.

L. Aversano, G. Canfora, A. De Lucia, and S. Stefanucci. 2002. Evolving ispell: A case study of program understanding for reuse. In *Proceedings of the 10th International Workshop on Program Comprehension*, p197. IEEE Computer Society.

B. Dömölki. 1967. Algorithms for the recognition of properties of sequences of symbols. *USSR Computational & Mathematical Physics*, 5(1):101–130. Pergamon Press, Oxford.

P. Halácsy, A. Kornai, L. Németh, A. Rung, I. Szakadát and V. Trón 2003. Creating open language resources for Hungarian. See LREC Proceedings.

Z. Harris. 1951. *Methods in Structural Linguistics*. University of Chicago Press, Chicago.

D. A. Hull. 1996. Stemming algorithms: a case study for detailed evaluation. *J. Am. Soc. Inf. Sci.*, 47(1):70–84.

C. Jacquemin. 1997. Guessing morphology from terms and corpora. In *Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval*, 156–165. ACM Press.

P. Kiparsky. 1982. Lexical phonology and morphology. In I.S. Yang, editor, *Linguistics in the Morning Calm*, 3–91. Hansin, Seoul.

W. Kraaij and R. Pohlmann. 1996. Viewing stemming as recall enhancement. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, 40–48. ACM Press.

J. B. Lovins. 1968. Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 11:22–31.

E. Nida. 1949. *Morphology: The Descriptive Analysis of Words*. University of Michigan, Ann Arbor.

C. D. Paice. 1994. An evaluation method for stemming algorithms. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, 42–50. Springer-Verlag New York, Inc.

J. L. Peterson. 1980. *Computer programs for spelling correction: an experiment in program design*, volume 96.

M. F. Porter. 1980. An algorithm for suffix stripping. *Program*, 14(3):130–137.

J. Xu and W. B. Croft. 1998. Corpus-based stemming using cooccurrence of word variants. *ACM Trans. Inf. Syst.*, 16(1):61–81.