A spellchecker for Afrikaans, based on morphological analysis

Gerhard B van Huyssteen *(Potchefstroom University for Christian Higher Education, Potchefstroom, South Africa) &* ***Menno M van Zaanen*** *(University of Amsterdam, Amsterdam, The Netherlands)*[1]

1. Introduction

Existing commercially available spellcheckers for Afrikaans (i.e. *PUK/Microsoft Speltoetser*; *Pharos Speller* (and hyphenator); *Ispell vir Afrikaans)* still don't comply with the general desiderata for spellcheckers (i.e. user-friendly, technically elegant, and linguistically justified, that gives the user high recall, high precision, and adequate suggestions), especially with regard to their lack in efficiency to cope with the structure and nature of Afrikaans as a semi-agglutinative language that allows for productive compound formation. This project under discussion is aimed at the development of an enhanced spellchecker for Afrikaans, which will not only improve in functionality and performance on existing Afrikaans spellcheckers, but also reach the benchmarks set by other comparable projects (e.g. the SCARRIE project[2]; see also Van Huyssteen, 2002). The three main areas of improvement are **coverage/recall** (i.e. the recognition of valid words as valid, and of errors as errors), **flagging/precision** (i.e. the correct indication of (potential) errors), and **suggestion adequacy** (i.e. to make useful suggestions for correcting errors) (Paggio & Underwood, 1995: 6-8). The project commenced in February 2002[3], and a first version of the new spellchecker will be released in July 2003. However, the project will continue after that, until the benchmark results are reached.

We will commence by discussing the general architecture of the spellchecker under development. The main focus of this paper will be on the evolution of the architecture of the spellchecker and the problems we are currently experiencing. We will then briefly discuss one of the morphological analysis modules, indicating the implications of this module on the spellchecker's recall and precision. As there is currently no final product or prototype available, results presented in this paper is preliminary and to a large extent inconclusive.

2. General Architecture

The spellchecker under development consists mainly of two kinds of modules, viz look-up modules and rule-based morphological analysis modules. The reusability of and the possibility to upgrade/refine these modules are constantly kept in mind during the development and integration phases.

2.1 Look-up modules

In the architecture, two look-up modules are mainly employed, viz a simple lexicon look-up module, and an error-detection module.

2.1.1 Lexicon look-up module

Although the ideal was initially set to reduce the size of the lexicon dramatically, it was later decided to increase the lexicon for two reasons: (a) in order to gain a few percentage points on lexical recall, and (b) to intercept problems that might arise during morphological analysis. An increased lexicon should, however, not be problematic, due to the size and speed of modern computers (the spellchecker is developed for Office XP®, and it is considered a given that end-users who use Office XP® will have rather powerful machines), as well as the efficiency of modern data structures. The lexicon of the spellchecker therefore currently consists of circa 250 000 items, including ordinary words, proper names, specialised vocabulary (e.g. chemistry and biology terms), frequent compounds, etcetera.

2.1.2 Error-detection module

Initially the architecture provided only for the lexicon look-up module, but, because of problems arising with the morphological analysis later in the spellchecking procedure, it was deemed necessary to add an error-detection module. This module consists of two parts: (a) a look-up section that contains a list of frequently misspelled words, and (b) a section where errors are detected based on 4-gram analysis at grapheme level.

2.1.2.1 Look-up section

The lexicon of the look-up section is based on a relatively small corpus of errors, collected via a web-based competition that was launched during the project (see www.puk.ac.za/lettere/spel2002), as well as from available electronic data (e.g. from e-mails, messages on the Potchefstroom University's bulletin board, etc.). When a string is found during the look-up procedure, the misspelled word with its correct form is send directly to the suggestion module. This lexicon will be extended and updated as more data becomes available.

2.1.2.2 4-gram analysis section

The 4-gram analysis section is based on a list of valid 4-grams for Afrikaans. To identify valid 4-grams, a software application was developed in Java, which enabled us to extract n-grams (i.e. n-graphs) from a corpus. The software also has the function to differentiate between n-grams that occur at the beginning, the end, and in the middle of words. As hyphens are not treated as word boundary markers during tokenisation at the start of the spellchecking session, and as hyphens can be used quite frequently in Afrikaans compounding, we had to accommodate hyphens in our lists of 4-grams. This procedure resulted in a list of 129 374 valid 4-grams, which are stored as three different sections of the 4-gram list.

Although the effectiveness and/or efficiency of this error-detection module can not be precisely measured at this stage of the project, we believe that early error-detection will save on processing time, and also result in higher suggestion adequacy. It will however be essential to thoroughly evaluate this module in the final stages of the project (i.e. when a prototype of the spellchecker is available). For example, we would certainly experiment with n-grams, deciding whether we should rather use trigrams or 5-grams. It could also be interesting to compare the overall results of the spellchecker with, without, or even only with the error-detection module.

2.2 Rule-based morphological analysis modules

In order to handle morphologically complex words (like compounds, derivations, derived compounds, etcetera) a few rule-based morphological analysis modules are build into the spellchecker. These modules include a stemmer, a word segmenter, and a module that combines word segments in different sequential strings.

After a string was not found in the lexicon look-up module, and was judged by the 4-gram analysis module to be a valid string, the string is sent to a stemmer. The stemming procedure will be discussed in more detail later (see 3.).

If the string is still not found, it is analysed by the rule-based word segmentation module, where a string is segmented according to the word segmentation principles of Afrikaans. After the string has been segmented, the lexicon look-up procedure is applied to all the resulting strings. If all forms are found, the spellchecking session ends; if not, and if a word consists of three or more segments, the segments are sent to the module that combines word segments in different sequential strings (i.e. the de-segmentation module). In this module the different segments are glued together step by step and in different combinations. Each of these combinations are looked-up in the lexicon. The session ends if the string is found. If not, the string is sent to the suggestions module, where suggestions are made based on an edit distance algorithm.

3. Module: stemmer

Stemming algorithms are usually employed in Information Retrieval (IR) environments (see Porter, 1980), and the aim of such stemmers is therefore to lump together “nonidentical words which refer to the same principal concept”, irrespective of whether the resulting stem is a “linguistically correct lemma or root” (Paice, 1990). In our case (i.e. with regard to a spellchecker) exactly the opposite holds true, where a linguistically correct form is more important than the semantics of the resulting stem.

The stemming algorithm that we are developing is based by and large on the design of the Porter Stemmer for Dutch (PSD) (cf. Kraaij & Pohlmann, 1994). For instance, like in the PSD, our stemming procedure is mainly based on affix stripping, and also includes some special conditions to cover certain phenomena (like the DupV-procedure, where closed syllables are identified and the vowel is subsequently doubled – see Kraaij & Pohlmann, 1994: 170-171). Similarly, we include a measure condition, as well as some clean-up rules to render valid stems.

However, considering the different aims of the two algorithms, our stemmer also differs in some ways from the PSD. In the PSD only "derivational affixes which do not substantially affect the information conveyed by the term" (Kraaij & Pohlmann, 1994: 170) are removed, whereas in our case, we are not restricted in this way. For example, the prefix *on–* ('un–') should, for semantic reasons, not be removed in an IR environment, whereas it can be harmlessly stripped off in a spellchecker. This entails that we do not have to restrict our algorithm to only the most frequent affixes (like in the PSD), but that we can also include less frequent affixes, resulting in a longer list of derivational affixes that are removed by our stemmer than that of the PSD. However, in order to preserve the efficiency of the stemming algorithm, we constrain ourselves to only include less frequent affixes that are highly regular (i.e. that combine only with free stems, or with a limited number of bounded stems), and do not cause any over-stemming problems. With regard to derivational affixes, our stemmer covers all affixes that are traditionally considered to be derivational affixes (see Jenkinson, 1993).

Another (slight) difference between the two stemmers has to do with the grouping and ordering of rules. In the PSD, the stemming rules (including the clean-up rules) were clustered into six groups in order to accommodate the level at which the affixes occur in the word formation process (Kraaij & Pohlmann, 1994: 170). In our stemmer, the rules are clustered in two main groups, viz. in an inflection stemmer, and a derivational stemmer. Within each of these two groups, rules are carefully grouped together and ordered according to their formal and functional behaviour (e.g. all the plural suffixes are grouped together, and the past participle prefixes/infixes are ordered according to the length of the string). This clustering allows us to use differentiated and apposite procedures for each of the categories: for example, the procedure that verifies and, if necessary, removes the "*d*" or "*t*" after the plural "*–e*" rule has fired (e.g. *gaste* 'guests' -> *gast* -> *gas* 'guest'), needs not to apply after the diminutive rules have fired.

Despite these ordering and clustering, we still run into a considerable amount of problems, especially with regard to over-stemming. For example, the algorithm states that, after removing the comparative "*–er*", also remove the potentially redundant "*t*" (*sagter* 'softer' -> *sagt* -> *sag* 'soft', where *sagt* is a bounded stem, originating from Dutch). If this algorithm is applied consistently, a word like *briljanter* ('more brilliant') is wrongly reduced to **briljan*. To solve this problem it seems as if we have two options: either include bounded stems in a special lexicon (thus *sagt* will be found, and **briljan* not), or add a look-up procedure after each rule that fires (thus the "*t*" of *briljant* will not be removed, because the string will be validated after *–er* is removed).

Although the former seems at first like the better, more "economic" option, it is also problematic. To compile a lexicon of bounded stems will be a very difficult, labour-intensive and time-consuming process. Moreover, even a carefully crafted lexicon of bounded stems can result in wrong judgements: if someone would type **sagties* (instead of *saggies* 'softly'), the resulting stem *sagt* will be judged a valid stem after the suffixes *–ie–s* are removed. Of course, the same problem will occur in the second option if one does not deal carefully with the linguistic reality. For instance, if one applies the "*d/t*" removal rule non-discretionary, **sagties* will also be judged a valid string. However, in Afrikaans the potentially redundant "*d/t*" does not occur with diminutive suffixes, and one could therefore specify that this rule should not apply after the diminutive rules have fired[4]. As it seems like the more linguistically justifiable option, we therefore include look-up procedures in the stemming algorithm, where necessary.

A major concern at this stage is what the influence of this stemming algorithm will be on the overall processing speed of the spellchecker. Given that one of our main aims is to improve lexical recall (and not to increase the speed of the spellchecker), we have decided to continue along these lines. If deemed necessary in the evaluation phase, adjustments will be made to this algorithm (e.g. by restricting the stemming to only certain affixes).

To conclude: more than 200 rules are employed in the stemmer to handle a set of frequently (as well as less frequently) occurring inflectional and derivational affixes. Although our stemmer has not been tested and evaluated thoroughly and systematically (and neither has it been fine-tuned), it yielded 85% correct stems in a preliminary test conducted on a 1000 word sample. As an 85% success rate is probably not good enough for use in a spellchecker, further work will continue to increase the performance and efficiency of the stemming algorithm.

4. Conclusion

Although we are at this stage of the project not able to reach final conclusions, it seems as if simple morphological decomposition could cause some unwanted analyses, which could lower the precision and the processing efficiency of the spellchecker. However, by introducing additional techniques and other measures, these problems could be minimised. For instance, instead of using word segmentation and documentation modules, one could rather introduce a longest string match algorithm, or one could prevent over-stemming mistakes by limiting the stemming algorithm to only the most frequent affixes. We are also considering using a Part of Speech tagger to prevent mistypings like *dieman* ('the+man') to be analysed as a valid string during word segmentation. These and other techniques will be explored further in the current project.

5. References

Jenkinson, A.G. 1993. Die probleem van fleksie en afleiding in Afrikaans. [The problem of inflection and derivation in Afrikaans]. *South African Journal of Linguistics Supplement*. 18: 100-122.

Kraaij, W. & Pohlmann, R. 1994. Porter's stemming algorithm for Dutch. In: Noordman, L.G.M. & De Vroomen, W.A.M. (eds.). *Informatiewetenschap 1994: Wetenschapelijke bijdragen aan de derde STINFON Conferentie*. pp. 167-180.

Paice, C.D. 1990. Another stemmer. *ACM-SIGIR Forum*. 24(3): 56-61.

Paggio, P., & Underwood, N.L. 1995. Validating the TEMAA LE evaluation methodology: a case study on Danish spelling checkers. *Natural Language Engineering*. 1(1): 1-18.

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

Povlsen, C., Hein, A.S. & De Smedt, K. 1999. *SCARRIE project: Final Project Report*. www.ling.uib.no/~desmedt/scarrie/final-report.html.

Van Huyssteen, G.B. 2002. Desiderata of spellchecking/spell-checking/spell checking: towards an intelligent spellchecker for Afrikaans. Paper presented at a one-day symposium, *Developing Spelling Checkers for South African Languages*. 14 March. Potchefstroom University for CHE, Potchefstroom, South Africa.

Vosse, T.G. 1994. *The Word Connection: Grammar-Based Spelling Error Correction in Dutch*. Ph.D. thesis. Leiden: Rijksuniversiteit Leiden.

---

[1] We thank and acknowledge the following members of the research team for their inputs in this research: Roald Eiselen, Christo Els, Petri Jooste, Christo Muller, Sulene Pilon, Martin Puttkammer, Werner Ravyse, and Daan Wissing. We would also like to express our gratitude towards Attie de Lange, Ulrike Janke, Boeta Pretorius, and Elsa van Tonder for technical, administrative, and legal support. The Potchefstroom University for CHE also sponsors this project generously – our thanks to Frikkie van Niekerk for his support.

[2] The spellcheckers developed in the SCARrie project is based on the design and architecture developed in the CORrie project, a project aimed at the development of a morphological-based spellchecker for Dutch (Vosse, 1994). As the SCARrie project improved on the CORrie project, the results obtained in the SCARrie project is set as the benchmark for the current project.

[3] This project has its seat at the Potchefstroom University for CHE (South Africa). In South Africa, the fields of Natural Language Processing and Computational Linguistics are by and large unexplored territory. However, since 2001 the Potchefstroom University prioritised these fields as strategically important research terrains, and invested extensively in the enhancement of personnel and other resources to develop human language technologies for some of the South African languages. As the IT department at the Potchefstroom University had already developed a spellchecker for Afrikaans (the *PUK/Microsoft Spellchecker*), the development of an enhanced spellchecker was considered an “easy” project to get hands-on experience in the above-mentioned fields.

[4] Of course, the –*ie* suffix in *saggies* is not a diminutive suffix, but it has the same form as the regular –*ie* diminutive suffix in words like *boekie* ‘booklet’ or *plekkie* ‘small place’.