Tools for Fast Morphological Analysis Based on Finite State Automata

Pavel Šmerk

Natural Language Processing Centre
Faculty of Informatics, Masaryk University
Botanická 68a, 602 00 Brno, Czech Republic
smerk@fi.muni.cz

Abstract.
The paper presents a new implementation of some of Jan Daciuk's algorithms and tools for morphological analysis based on finite state automata [1]. In particular, we offer a reimplemented version of the tool which builds the automata from an input set of strings and of the tool which performs the morphological analysis itself. In addition to 8-bit versions we also offer "Unicode-aware" versions with the Unicode characters encoded directly in the arcs of the automaton. The new implementation is faster than the original one and its code is much more simple and straightforward.

Keywords: morphological analysis, minimal deterministic finite state automata

1 Introduction

Computational morphological analysis is one of the first steps in the automatic treatment of natural language texts. Just after splitting the processed text into words we usually need a tool which for each such word returns its possible corresponding lexical entries (*lemmata*) and a relevant grammatical information. For languages with limited compounding and with morphology realized mainly by changes at the end of the word (Slavic languages can be taken as an example), it is possible to describe their morphology by means of a simple list of all words (word forms) and their possible interpretations and to let the morphological analyzer only search this list for each input word.

Of course, it would not be feasible to search such a list directly due to its huge size. However, the list can be viewed as a finite (formal) language and consequently it can be represented by a minimal deterministic acyclic finite state automaton, which can be pretty small. The morphological analyzer then only follows a path in the automaton according to letters of the analyzed word and, if a corresponding path exists, returns all possible continuations of this path as a result.

In the following section we describe the input data format and illustrate how the morphological analysis works. In the next section we present results

Aleš Horák, Pavel Rychlý (Eds.): Proceedings of Recent Advances in Slavonic Natural Language Processing, RASLAN 2014, pp. 147–150, 2014.

of newly reimplemented tools and finally we discuss some possible future modifications.

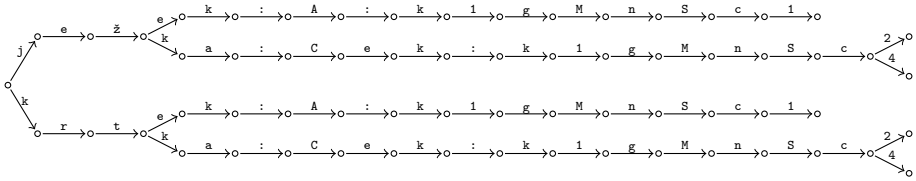
2 Data for morphological analysis

The data for morphological analysis are simply a list of all combinations of recognized input strings and corresponding outputs of the analyzer, where pairs of two words are encoded as pairs formed by the first word and a difference between the words [2]. For example, in the following part of data for word form → lemma + tag analysis (with the original data on the right side and the encoded form on the left side)

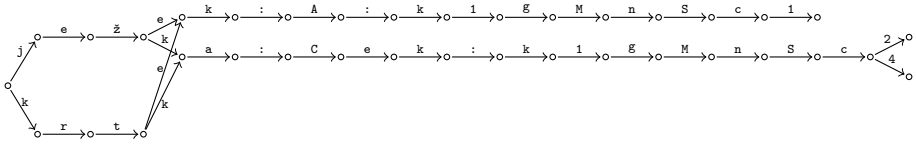
```
ježek:A:k1gMnSc1      ← ježek:ježek:k1gMnSc1
ježka:Cek:k1gMnSc2    ← ježka:ježek:k1gMnSc2
ježka:Cek:k1gMnSc4    ← ježka:ježek:k1gMnSc4
krtek:A:k1gMnSc1      ← krtek:krtek:k1gMnSc1
krtka:Cek:k1gMnSc2    ← krtka:krtek:k1gMnSc2
krtka:Cek:k1gMnSc4    ← krtka:krtek:k1gMnSc4
```

the (first) colon is a delimiter between the possible inputs and corresponding outputs and the letters A and C as the first and the third letters of the alphabet mean "to get the lemma delete n-1 (i.e. 0 or 2, respectively) last characters from the word form and then attach the rest of the string (i.e. empty string or `ek`, respectively)". For example, the word form `krtek` will be analyzed as a lemma `krtek` with a morphological tag `k1gMnSc1`[1] and a word form `krtka` as a lemma `krtek` with morphological tags `k1gMnSc2` or `k1gMnSc4`[2].

Such a list is then represented as a minimal deterministic acyclic finite state automaton using Jan Daciuk's algorithms for incremental building of minimal DAFSAs [1]. The following graph corresponds to the non-minimized automaton (trie)



and the second graph corresponds to the minimized automaton used for the morphological analysis.



[1] *mole* in nominative form

[2] *mole* in genitive and accusative form

This representation dramatically reduces the size of the data (some particular figures can be seen later in Table 1). The lookup is then very simple: if the analysed string concatenated with the delimiter is found in the automaton, then each possible remaining path to a final state of the automaton encodes one of possible analyses.

It means that there is no "real" analysis as any sophisticated algorithm above some grammar model or a system of paradigms, but whole analysis is only a simple — and therefore fast — dictionary search.

3 Experiments and results

3.1 Building the data for morphological analysis

We demonstrate our results on four sets of data. English and Russian morphological data are from the project FreeLing, the data for Czech are ours. We compare our new implementation with the original Daciuk's implementations[3] and with Java reimplementation of David Weiss[4] from project Morfologik (it also offers a more compact format at a price of a greater build time, for details refer to [3]). For the purpose of comparison, our implementation produces binary identical output (except for custom header) as the original Daciuk's fsa_build built with `-DFLEXIBLE -DNEXTBIT -DSTOPBIT` compile options. The first table describes the data sets and in the last column is the size of the resulting automaton (both input and output sizes are in bytes).

Table 1: Data sets used in the experiments.

data set	input size	words (lines)	output size
EN	1,417,920	88,652	244,764
RU	114,605,988	2,844,516	3,639,960
CZ free	105,001,670	3,393,080	931,594
CZ full	828,973,970	27,764,093	3,795,423

The second table presents build times for the three compared tools.

Table 2: Build times in seconds.

data set	fsa_build	morfologik	new implem.
EN	0.24	0.59	0.08
CZ free	12.63	7.50	4.19
RU	26.04	10.19	9.41
CZ full	121.41	57.21	41.71

[3] `www.eti.pg.gda.pl/katedry/kiw/pracownicy/Jan.Daciuk/personal/fsa.html`, version 0.51

[4] `http://sourceforge.net/projects/morfologik/`, version 1.9.0

3.2 Morphological analyzer and UTF-8 versions

We also offer a new implementation of morphological analyzer. Unfortunately, we were not able to make fsa_morph case conversions work even with Daciuk's original language files, thus it is difficult to compare analysis times in real world scenarios. If we changed our analyser to immitate (somewhat broken) fsa_morph output for 10 million Czech words from corpus, we are ca. 20 % faster (12.51 s $\times$ 15.25 s).

Daciuk's tools allow to be compiled with UTF-8 support, but it requires the user to describe the case conversions and diacritics adding/removal. We have UTF-8 variants of our tools which works with automata with UTF-8 labels and our case conversions and diacritics restoration follows the Unicode standard, which means one solution for all languages.

Except for speed, another advantage of our solution is much shorter and straighforward code. It is not easy to make a fair comparison, but, for example, files needed for fsa_build have more than 200 kB in total, whereas the code of our new implementation has less than 20 kB. It is obvious that in case of such a huge difference it is easier to maintain, adjust and further develop the shorter code.

The new tools are accessible from `http://nlp.fi.muni.cz/ma`.

4 Future work

The new tools are ready to use and the presented results are promising, but it is still a work in progress. We plan to further reduce both time and final size of the automata construction. We want to employ some variable length encoding of unicode codepoints, numbers and addresses (similar to [1], but computationally simpler one). We suspect Daciuk's "tree index" used to discovering already known nodes during the automaton construction to be slow for large data and we hope that simple hash will decrease the compilation time significantly.

Acknowledgements This work has been partly supported by the Ministry of Education of CR within the Lindat Clarin Center LM2010013.

References

1. Daciuk, J.: Incremental Construction of Finite-State Automata and Transducers, and their Use in the Natural Language Processing. PhD thesis, Technical University of Gdańsk, Gdańsk (1998)
2. Kowaltowski, T., Lucchesi, C.L., Stolfi, J.: Finite Automata and Efficient Lexicon Implementation (1998) Technical Report IC-98-02, University of Campinas, São Paulo.
3. Daciuk, J., Weiss, D.: Smaller representation of finite state automata. Theoretical Computer Science **450**(0) (2012) 10–21