Optimising Morphological Analyser Transducers Using Morphological Structure and Automatically Induced Flag Diacritics - Abstract

Senka Drobac[1], Krister Lindén[1], Tommi A Pirinen[2] and Miikka Silfverberg[1]

[1]Department of Modern Languages, PO Box 24, 00014 University of Helsinki
[2]Department of Speech Sciences, PO Box 9, 00014 University of Helsinki

1 Rationale

Flag diacritics enable optimising finite-state networks by combining identical sub-graphs of its transition graph. Traditionally, the feature has required linguists to devise the optimisations to the graph by hand alongside the morphological description. In this paper, we present a novel method of discovering flag positions in morphological lexicons automatically based on the morpheme structure implicit in the language description. With this approach, we have gained significant decrease in the size of finite-state networks while maintaining reasonable application speed.

2 Introduction

Finite-state transducers are an established way of encoding morphological analysers for natural languages. Nevertheless, full-scale morphological analysers can often grow to be too large for use cases like spell checkers, speech processing and shallow parsing, which should have moderate memory footprint. Large transducers can be optimised by recognising equivalent sub-graphs and combining them by using special symbols called flag diacritics, to couple entrance points of the sub-graphs with the correct exit points.

Up to date, applying flag diacritics has required a linguist to provide the lexicon compiler with their positions. However, there are two major problems with this kind of approach: Firstly, linguists often do not have a very good understanding of the structure of the finite-state networks built from lexicographical-morphological descriptions; Secondly, the addition of flag diacritics to these descriptions makes them unreadable and unmanageable since the amount of non-linguistic data in the linguistic description increases.

One of the reasons why flag diacritics have been so cumbersome from the linguists point of view, is their two-fold nature. On one hand, they are there to optimise the finite-state automaton structure, e.g. in (Karttunen, 2006). On the other hand, they are the primary method of describing non-contiguous morphological constraints (Beesley, 1998). If they are applied on the constriction of separated morphotactic dependencies, the effect on optimisation is at best haphazard, and the resulting description is neither linguistically motivated nor maintainable from a computational view-point.

This article seeks to address problems associated with flag diacritics by using an algorithm for inducing flag positions from the linguistic morpheme structure, implicitly present in lexical descriptions.

3 Background

Finite state morphology (Beesley and Karttunen, 2003) is the state-of-the-art in writing morphological analysers for natural languages of the whole range of typologically varying morphological features. The finite-state approach is built around two practical concepts: constructing lexicographical descriptions of the language using a syntax called lexc and expressing morphophonological variations as regular expression rules. In this paper we study the use of lexicographical structure as framed by lexc.

Lexc is a simple right-linear phrase structure grammar formalism. In linguistic terms this means approximately the following: we have collections of lexicons, which are lists of morphemes. Each morpheme in a lexicon defines a continuation lexicon, which in turn determines the set of morphemes that can succeed the original morpheme and their possible continuations.

Consider for example Finnish morphology. Nominal inflection can be constructed neatly from left to right. In figure 3, there is a lexc representation of the Finnish words *talo* ‘house’, *asu* ‘clothing’ and *kärry* ‘cart’, and nominal suffixes

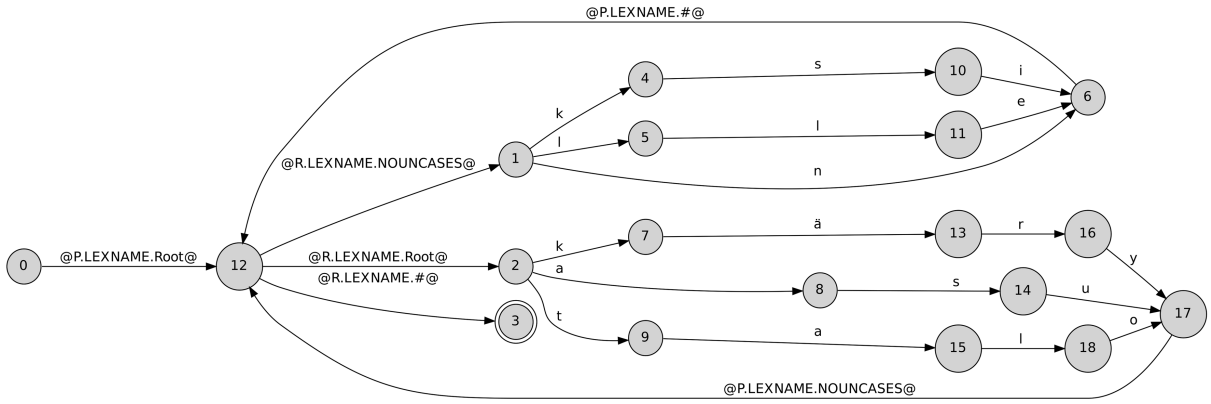

Figure 1: Simplified part of Finnish lexc grammar description with automatic flags

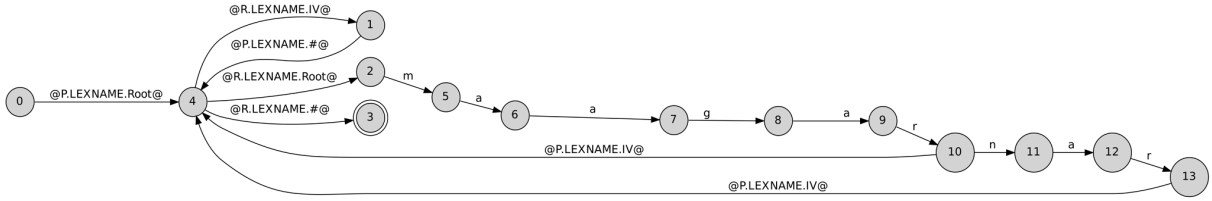

Figure 2: Simplified part of Greenlandic lexc grammar description with automatic flags

```
LEXICON Root
talo NOUNCASES ;
asu NOUNCASES ;
kärry NOUNCASES ;

LEXICON NOUNCASES
n # ;
lle # ;
ksi # ;
```

Figure 3: Simplified part of Finnish lexc grammar description

n (singular genitive), *lle* (singular allative) and *ksi* (singular translative). Derivation of word-forms starts from the `Root` lexicon. Each of the nouns in root set of morphemes continues rightwards to `NOUNCASES` set of morphemes, and each case morpheme continues towards the special `#` lexicon signifying the end of a word-form.

Finnish was used as an example in Karttunen's paper on flag diacritics in optimisation (2006). In that paper, he showed that the optimisation quality of wisely selected flag diacritics can be substantial; from a 20,498 state automaton to an 1,946 state one. The article describes Finnish numerals, which have the feature of requiring agreeing inflection in free compounding. This can be achieved by allowing all compounds and restricting the combinations by flags, instead by lexicon structure. Unfortunately, the article does not show examples or reproducible description of the lexicographical data, but to our experience there are no available morphologies that show similar compression quality, so it can be considered towards the upper bounds of what such compression can achieve.

4 Methods

Flag diacritics are special multi-character symbols which are interpreted during runtime. They have special syntax: `@operator.feature.value@`, where `operator` is one of the available operators (P, U, R, D, N, C), `feature` is name of a feature provisionally set by user and `value` can be any value held in feature, also provisionally defined (Beesley and Karttunen, 2003).

In this paper, we will use only two types of flag diacritics: positive setting (`@P.feature value@`) and require test (`@R.feature value@`). While positive setting flag only sets the feature to its value, require test flag invokes testing whether the feature is set to the designated value. For example, `@P.LEXNAME.Root@` will set feature `LEXNAME` to value `Root`. If later in the path there is an R flag that requires test `@R.LEXNAME.Root@`, the in-

voked test will succeed and that path will be considered valid.

Our algorithm is based on the finding that adjacent morph combinatorics can be expressed with finite-state flags like this:

Every rightward continuation is replaced with a positive setting flag with feature called `LEXNAME` and value corresponding to the continuation lexicon. For example, the lexicon in figure 3 has two rightward continuations: `NOUNCASES` and `#`, which are represented using flags `@P.LEXNAME.NOUNCASES@` and `@P.LEXNAME.#@`. Similarly, every leftward continuation is replaced with request test flag which also has feature `LEXNAME` and the corresponding value. Therefore, the lexicon shown in figure 3 will also have two leftward continuations: `NOUNCASES` and `#`, which are represented using flags `@R.LEXNAME.NOUNCASES@` and `@R.LEXNAME.#@`. Additionally, every morphological description starts with `Root`, which is represented using the pair of flags `@P.LEXNAME.Root@@R.LEXNAME.Root@`.

The transducer built from the morphological description in figure 3 is shown in figure 1.

Since real-world morphological descriptions often contain empty, or nearly empty, continuation lexicons, inserting flags in those cases only increase size of the transducer, without gaining any benefits. Therefore, those continuation lexicons are recognized during compilation time and skipped when inserting corresponding flag diacritics.

Lexicons that contain flag diacritics can be composed with other transducers which also contain flag diacritics without worrying about flag collisions. This is achieved by renaming flag diacritics in both argument transducers in such a way that collisions become impossible and then inserting flag diacritics freely from each argument to the other.

Consider for example composition of a lexicon L with a rule R. If both transducers contain flag diacritics for feature `FEATURE`, then $allfeaturesF$ are renamed `F1` in L and `F2` in R.[1] All flag diacritics (like `@P.F.True@`) are renamed correspondingly (to `@P.F1.True@` in L and `@P.F2.True@` in R) and a new lexicon L' and rule R' are created by inserting freely all flag diacritics from R to L and from L to R.

The transducers L' and R' can be composed and it is easy to see that the result satisfies the property that, if flag diacritics are compiled out, then the resulting transducer without flag diacritics will accept exactly the same strings as the composition of the transducers that are obtained by compiling out flag diacritics from the original lexicon L and the original rule R.

Language	Original	With flags	%
Greenlandic	168	17	10,1%
North Saami	12	5,7	47,5%
Lule Saami	5	3	60,0%
Erzya	3,7	5,3	143,2%

Table 1: Sizes of transducers without and with automatic flags (in megabytes); Percentage shows size of the flagged transducer in comparison to the original

In table 1 there are shown sizes of the original transducers compiled with regular lexc compiler composed with grammar rules, transducers compiled with new method that inserts flag diacritics composed with grammar rules and finally their ratio.

5 Data

We measure the success of our algorithm using real-world, large scale language descriptions. For this purpose we have acquired freely available, open source language descriptions from University of Tromsa's language repository (Moshagen et al., 2013).[2] The languages selected are Greenlandic (kal), North Saami (sme), Erzya (myv), and Lule Sami (smj).

All operations with transducers were performed using Helsinki Finite State Technology tools (Lindén et al., 2011).

6 Discussion

The results of this study show that large scale language descriptions can be compiled into smaller transducers using automatically inserted flags. The effect is especially pronounced for language descriptions which repeat morphemes in many different places, like the morphological analyzer for Greenlandic. Since flag diacritics themselves take space in the transducer graph, this method did

[1] It is not sufficient to rename only flag diacritics with common features, because that might clash with existing feature names.

[2] https://victorio.uit.no/langtech, revision 73836

not offer improvements for descriptions where the original transducer was small.

While requiring R flag diacritics will always occur only once for every left continuation lexicon, the results have shown that, for certain right continuations, P flag diacritics occur hundreds of times. This will happen every time when in the same set of morphemes there are words which have the same beginning. For example, in figure 2 shows how two morphemes maagar and maagarnar have same continuation flag @P.LEXNAME.IV@, but they can't collapse into the same path. In future work, it should be checked if skipping inserting flags for those paths would further reduce size of transition graphs.

7 Conclusion

In this article we showed that by using morphologically motivated flags we can dramatically improve the size of the large transducers. Automatically inserted flag diacritics would make manual optimization preformed by linguists unnecessary, which would result in more readable and easier to maintain linguistic descriptions.

References

Kenneth R Beesley and Lauri Karttunen. 2003. *Finite state morphology*, volume 18. CSLI publications Stanford.

Kenneth R Beesley. 1998. Constraining separated morphotactic dependencies in finite-state grammars. In *Proceedings of the International Workshop on Finite State Methods in Natural Language Processing*, pages 118–127. Association for Computational Linguistics.

Lauri Karttunen. 2006. Numbers and finnish numerals. *SKY Journal of Linguistics*, 19:407–421.

Krister Lindén, Erik Axelson, Sam Hardwick, Tommi A Pirinen, and Miikka Silfverberg. 2011. Hfst—framework for compiling and applying morphologies. In Cerstin Mahlow and Michael Piotrowski, editors, *Systems and Frameworks for Computational Morphology*, volume 100 of *Communications in Computer and Information Science*, pages 67–85. Springer Berlin Heidelberg.

Sjur Moshagen, Tommi A Pirinen, and Trond Trosterud. 2013. Building an open-source development infrastructure for language technology projects. In *Proceedings of Nodalida 2013*. forthcoming.