Automated Lossless Hyper-Minimization for Morphological Analyzers

Senka Drobac and **Miikka Silfverberg** and **Krister Lindén**
Department of Modern Languages
PO Box 24
00014 University of Helsinki
{senka.drobac, miikka.silfverberg, krister.linden}@helsinki.fi

Abstract

This paper presents a fully automated lossless hyper-minimization method for finite-state morphological analyzers in Xerox `lexc` formalism. The method utilizes flag diacritics to preserve the structure of the original `lexc` description in the finite-state analyzer, which results in reduced size of the analyzer. We compare our method against an earlier solution by Drobac et al. (2014) which requires manual selection of flag diacritics and results in slow lookup. We show that our method gives similar size reductions while maintaining fast lookup without requiring any manual input.

1 Introduction

Morphological analyzers are commonly implemented as finite state machines (FSM) because finite-state technology enables both fast processing of large amounts of input and manipulation of the analyzer using finite-state algebra. Sometimes finite-state analyzers may, however, become quite large. This can be a problem e.g. when analyzers are used on mobile devices where a moderate memory footprint is required.

The usual way to reduce the size of FSMs is to use a minimization algorithm (Hopcroft, 1971). Minimization can have a substantial effect on the size of the FSM but, as it is only able to combine suffix-equivalent states, there may still be residual redundancy in the state space of the machine.

Further size reduction can be accomplished by introducing a limited form of context-free structure into the finite-state graph using special symbols called flag diacritics (Beesley, 1998). Using flag diacritics, it is possible to combine sub-graphs which are equivalent, i.e. accept the same strings, but which are not necessarily suffix-equivalent. Flag diacritics are used to couple entrance points of the sub-graphs with appropriate exit points. During lookup, paths whose flag diacritics do not match are filtered out. Thus, the original language of the machine is preserved.

Traditionally, the lexicon writer manually inserts flag diacritics into the lexicon of the morphological analyzer. There are two major problems with this approach: (1) In practice, manually inserted flag diacritics often do not result in great size reduction because many lexicon writers have poor understanding of the structure of the finite-state networks built from lexicographical-morphological descriptions; (2) 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.

This paper introduces an automated method for inducing flag diacritics into finite-state morphological analyzers based on the Xerox `lexc` formalism. We refine an earlier approach by Drobac et al. (2014), which requires manual selection of flag diacritics, to obtain substantial size reduction. We show that our approach achieves similar reductions in size, but with a fully automated process. Moreover, the approach presented in this paper is conceptually simpler and faster because, unlike Drobac et al. (2014), we do not need additional processing after applying phonological rules. Additionally, our approach results in substantially improved lookup speed compared to Drobac et al. (2014) due to an operation which we call path condensation.

We apply our approach to morphological analyzers for three morphologically complex languages: Greenlandic, North Saami, and Finnish. Compared to Drobac et al. (2014), our approach results in near equal size reduction for these languages without requiring any manual intervention. Furthermore, due to path condensation introduced in Section 3.3, lookup time is reduced for all three languages and because of the new `lexc` approach, compilation time is reduced for all languages.

2 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 a language by using a grammar formalism called `lexc` and expressing morphophonological variations as regular expression rules. In this paper, we study lossless hyper-minimization of finite-state machines derived from lexicographic descriptions in the `lexc` formalism.

In this paper, we use the term hyper-minimization of minimal deterministic finite-state machines to refer to procedures which produce an even smaller finite-state machine, that preserves some of the qualities of the original deterministic minimal machine. This definition of hyper-minimization is broad enough to encompass a number of different approaches. The generally used definition of hyper-minimization, introduced by Badr et al. (2009) and further developed by for example Maletti and Quarnheim (2011), is more restricted.

The hyper-minimization algorithms investigated by Badr et al. (2009) and by Maletti and Quarnheim (2011) introduce a limited amount of changes to the language accepted by the original machine. This makes the machine susceptible to further size reduction using conventional minimization algorithms. We call this kind of hyper-minimization *lossy hyper-minimization* because the resulting finite-state machine does not accept the same language as the original machine. Lossy hyper-minimization results in a deterministic machine which allows fast lookup.

In contrast to lossy hyper-minimization, the approach presented in this paper is *lossless*. We introduce a limited amount of non-determinism into the finite-state machine using labeled epsilon symbols (flag diacritics). This allows us to achieve a size reduction, while at the same time preserving the original language accepted by the finite-state machine. The non-determinism introduced by the algorithm results in some reduction in lookup speed, which is not prohibitive in practice.

```
LEXICON Root
0 N ;
0 Adj ;

LEXICON N
cat+N:cat Num ;

LEXICON Adj
small+A:small Deg ;

LEXICON Num
+Sg:0 # ;
+Pl:s # ;

LEXICON Deg
+Pos:0 # ;
+Comp:er # ;
+Sup:est # ;
```

Figure 1: A `lexc` lexicon

Finding the smallest non-deterministic finite-state machine equivalent to a given machine is PSPACE complete (Jiang and Ravikumar, 1993) for general finite-state machines and therefore intractable. Nevertheless, using a linguistic description rather than a compiled finite-state machine as the starting point, it is possible to achieve substantial reduction in the size of the machine without penalties in compilation time.

3 Methods

In this section, we describe our lossless hyper-minimization algorithm of morphological lexicons. The algorithm is applicable on finite-state lexicons formulated as regular grammars. An example of this kind of formalism is the Xerox `lexc` formalism (Beesley and Karttunen, 2003). Sections 3.1 and 3.2 closely follow Drobac et al. (2014) but path condensation presented in Section 3.3 represents new work.

In the lexc formalism, lexicons are formulated as right branching regular grammars of morpheme continuation classes as demonstrated in Figure 1. Each path through the grammar defines one word form and its analysis.

`lexc` lexicons are compiled into finite-state transducers, which accept exactly the set of correspondences between word forms (e.g. `cats`) and analyses (`cat+N+Pl`) defined by the regular grammar using well known methods (Hopcroft et al., 2006).

Our method is not based on transforming the grammar into a finite-state machine directly. Instead we utilize so called *lexicon joiners* introduced by (Lindén et al., 2009). Joiners are specialized labels (e.g. `$N$` and `$Deg$`) that are appended to each entry in the lexicon. They identify the sub-lexicon of the entry and its continuation class. Together with a set of finite-state constraints, this completely encodes the structure of the original grammar.

All regular expressions and regular rewrite rules in this paper use Xerox `xfst` formalism.

3.1 Compiling lexicons using joiners

Compiling a lexicon using joiners starts with appending appropriate joiners to each lexicon entry. E.g. the entry `cat+N:cat` with continuation class `Num` in sub-lexicon `N` becomes `$N$cat+N$Num$:$N$cat$Num$`. We then compile all the modified lexicon entries into one trie `T` and form its Kleene closure `T*`.

The closure `T*` accepts arbitrary combinations of morphs from the original lexicon. In order to restrict it to valid combinations produced by the original `lexc` lexicon, we

- append root and end joiners to `T*` and get a language `$Root$ T* $#$`, and
- apply one finite-state constraint `CJ` for each joiner `$J$` type, which requires that each `$J$` has to occur next to another identical joiner `$J$`. E.g.

  ```
  CJ = NOJ* [$J$ $J$ NOJ*]*
  ```

  where `NOJ = [?* - $J$]`.

These constraints encode the structure of the original lexicon. After composing the language `$Root$ T* $#$` and the joiner constraints, the resulting lexicon transducer `L` accepts exactly the word forms and analyses defined by the original regular grammar, though interspersed with joiner symbols. In a final processing state, the joiners are removed and the lexicon is determinized and minimized.

3.2 Hyperminimization using Flag Diacritics

When compiling lexicons using joiners, it is often the case that the lexicon is small before the joiners are removed and the lexicon is determinized and minimized but grows in size after this final processing stage. This may seem surprising, as the transducers essentially encode the same strings notwithstanding joiner symbols. However, joiner symbols seem to add useful structure into the lexicon which helps to maintain a smaller size.

By transforming each joiner `$J$` into a flag diacritic `@P.J.ON@`[1], we can maintain the useful structure while at the same time allowing for lookup of word forms and further processing such as application of rules and minimization. Note, that there is no actual flag diacritic functionality involved apart from the fact that the symbols are treated as labeled epsilon symbols. We use flag diacritics instead of custom made labeled epsilon symbols because flag diacritics are supported by a number of different finite-state toolkits.

3.3 Path Condensation

Although using joiners results in a smaller lexicon, it also slows down transducer lookup. Sometimes the slowdown can be rather drastic, even around 70% (Drobac et al., 2014). Our initial experiments showed that this happens mainly because of empty lexicon entries, which can result in long sequences of joiner symbols when chained together. During regular compilation, the chains of joiners vanish when all joiners are removed. In our case, however, the chains are unfortunately preserved.

In order to speed up lookup, we apply a final optimization stage before converting joiners into flag diacritics. In this stage we apply a parallel rewrite rule which transforms all sequences of joiner symbols into a single joiner symbol, i.e. "condenses" consecutive joiners into one joiner. Let `JOINER` be the set of of joiner symbols `[$J1$| ... | $JN$]`, then the rule (in Xerox `xfst` notation) is

```
JOINER -> 0 || _ JOINER ;
```

The rule and its inverse are composed with the output and input side of lexicon, respectively. Finally, the lexicon is determinized and minimized.

It is easy to see that path condensation preserves the original language encoded by the `lexc` lexicon disregarding joiner symbols.

4 Experiments

We performed experiments using three full scale open-source morphological analyzers distributed by the Giellatekno project[2]. We used the analyzers for Finnish (fin), Greenlandic (kal) and Northern Sami (sme) available from the Giellatakno repos-

[1] The flag diacritic `@P.J.ON@` sets the value `ON` for feature `J`.

[2] `http://giellatekno.uit.no/`

itory[3]. For compilation we use the Helsinki Finite State Transducer (HFST) library and tools[4] (Lindén et al., 2011).

We compile the morphological analyzers in three different ways

- Basic compilation without joiner symbols.
- Compilation using hyper-minimization.
- Compilation using hyper-minimization and path condensation.

For each compilation method, we report results on

- Compilation time.
- Size of the final morphological analyzer.
- Lookup speed of the final morphological analyzer.

Lookup speed is tested using continuous text spanning tens of thousands of words for each language. All experiments were performed on an Intel Core i5-4300U laptop with a dual core 1.90 GHz processor and 16 GB of RAM.

5 Results

The results of experiments are shown in Table 1.

Compilation time using hyper-minimization and path condensation does not seem to be prohibitive for any of the analyzers. In fact compilation time for both Greenlandic and Northern Sami is reduced by over 60% compared to compilation without hyper-minimization. The compilation time for Finnish, however, increases compared with compilation without hyper-minimization.

Hyper-minimization together with path condensation decreases the size of the Greenlandic lexicon by over 90% from 140 MB to 13 MB. The size of the Northern Sami transducer also decreases by approximately 7% but hyper-minimization does not seem to have an appreciable effect on the Finnish analyzer. These results are almost as good as the results obtained by Drobac et al. (2014) who report size reduction of 90% for Greenlandic, 17% for Northern Sami and 6% for Finnish.

Lookup speeds for hyper-minimized transducers with path condensation are greatly improved to at least 77 % of the original lookup speed from around 30 % of the original speed using only hyper-minimization without path condensation. These are much better than the lookup speeds reported by Drobac et al. (2014), where the lookup speeds for hyper-minimized Finnish and Nortern Sami dropped to below 40 % of the original lookup speeds.

Finnish

	None	H-M	H-M + PC
Compile (min)	1	6	5
Size (MB)	18	17	18
Lookup (kw/s)	103	31	79
Speed of orig.	100%	30%	77%

Greenlandic

	None	H-M	H-M + PC
Compile (min)	6	2	2
Size (MB)	140	12	13
Lookup (kw/s)	2	2	2
Speed of orig.	100%	100%	100%

Northern Sami

	None	H-M	H-M + PC
Compile (min)	7	1	1
Size (MB)	14	13	13
Lookup (kw/s)	39	13	31
Speed of orig.	100%	33%	79%

Table 1: Results of experiments. The columns denote (1) compilation without hyper-minimization (None), (2) with hyper-minimization (H-M) and (3) hyper-minimization together with path condensation (H-M + PC). The rows denote (1) compilation time, (2) fst binary size, (3) fst lookup speed (as thousands of words per second) and (4) lookup speed compared with the original compiled transducer.

6 Discussion and Conclusions

Although comparison between our method and (Drobac et al., 2014) is not entirely fair, because our method is fully automatic, and their method allows manual selection of joiner flags, it is clear that our results are similar with regard to size of the analyzers. Lookup speed, however, is greatly improved because of path condensation.

Removal of some manually selected flag diacritics following Drobac et al. (2014) would probably

[3] `svn co -r 109628 https://victorio.uit.no/langtech/trunk main`

[4] `hfst.sf.net`

give us an even smaller binary size while maintaining high lookup speed.

7 Acknowledgements

We want to thank the anonymous reviewers for their useful comments and suggestions.

References

Andrew Badr, Viliam Geffert, and Ian Shipman. 2009. Hyper-minimizing minimized deterministic finite state automata. *RAIRO-Theoretical Informatics and Applications*, 43(01):69–94.

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.

Senka Drobac, Krister Lindén, Tommi A Pirinen, and Miikka Silfverberg. 2014. Heuristic hyper-minimization of finite state lexicons. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland, May.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2006. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

John Hopcroft. 1971. An n log n algorithm for minimizing states in a finite automaton. Technical report, DTIC Document.

Tao Jiang and B. Ravikumar. 1993. Minimal nfa problems are hard. *SIAM J. Comput.*, 22(6):1117–1141, December.

Krister Lindén, Miikka Silfverberg, and Tommi Pirinen. 2009. Hfst tools for morphology an efficient open-source package for construction of morphological analyzers. In Cerstin Mahlow and Michael Piotrowski, editors, *State of the Art in Computational Morphology*, volume 41 of *Communications in Computer and Information Science*, pages 28–47. Springer Berlin Heidelberg.

Krister Lindén, Erik Axelson, Sam Hardwick, Tommi A Pirinen, and Miikka Silfverberg. 2011. HFSTFramework 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.

Andreas Maletti and Daniel Quernheim. 2011. Optimal hyper-minimization. *Int. J. Found. Comput. Sci.*, 22(8):1877–1891.