Proceedings of Symposium in Information and Human Language Technology. Natal, RN, Brazil, November 4–7, 2015.

JCLexT: A Java Tool for Compiling Finite-State Transducers from Full-Form Lexicons

Leonel F. de Alencar, Philipp B. Costa, Mardonio J. C. França, Alexander Ewart, Katiuscia M. Andrade, Rossana M. C. Andrade*

Group of Computer Networks, Software Engineering, and Systems (GREat) – Universidade Federal do Ceará (UFC)
Campus do Pici, Bloco 942-A – CEP 60455-760 – Fortaleza – CE – Brazil

jclext@great.ufc.br

Abstract. *JCLexT is a compiler of finite-state transducers from full-form lexicons, this tool seems to be the first Java implementation of such functionality. A comparison between JCLexT and Foma was performed based on extensive data from Portuguese. The main disadvantage of JCLexT is the slower compilation time, in comparison to Foma. However, this is negated by the fact that a large transducer compiled with JCLexT was shown to be 8.6% smaller than the Foma created counterpart.*

1. Introduction

Finite-state transducers (FSTs) have been the preferred devices used to implement morphological parsers (Trommer, 2004). They store information in a compact manner and allow for quick lookup times, being superior to concurring alternatives.

Computing is increasingly moving towards mobile platforms. Due to this, the need for natural language processing tools which are designed for the specifics of this setting has emerged. Alencar et al. (2014), for example, addresses this issue with the proposal of JMorpher, a finite-state morphological parser in Java able to natively run on Android devices. However, the JMorpher tool is limited in its functionality, as it was designed to apply an existing finite-state transducer to an input text.

JCLexT aims to resolve this limitation, it is a Java tool that can compile a full-form lexicon into a finite-state transducer. JCLext emulates the `read spaced-text` command found in XFST (Beesley and Karttunen, 2003), Foma (Hulden, 2009) and HFST (Lindén, Silfverberg and Pirinen, 2009). It is the first Java tool of this sort that we are aware of, furthermore it was not based on any existing implementation.

JCLexT was inspired by the minimization algorithm for acyclic deterministic automata (DFSA) proposed by Revuz (1992). As a pure Java implementation, JCLexT inherits the advantages of this programming language, this includes better portability, such as being able to run on Android, desktop, servers or as a web service with minor changes to the existing implementation. Furthermore being written in Java keeps JCLexT platform compatible with the existing JMorpher software.

The main purpose of this work was to emulate and improve on the existing functionality of Foma, with the goal of trying to achieve better results with very large full-form lexicons.

* R. M. C. Andrade is a CNPq Research Fellow, DT Level 2. For invaluable contributions, we are grateful to Priscila Sales, Kleber Bernado, and Pedro Belmino.

It is worth noting as a result of this work we created LEXPT01, a lexical transducer of Portuguese which contains approximately 4 million paths. As far as we know, this is the largest lexical transducer of Portuguese currently in existence.

2. Algorithmic issues

In this paper, we skip the details of the main algorithm that underlies JCLexT. This algorithm is responsible for compiling a full-form lexicon into an FST, this has a reduced size in comparison to a baseline resulting from the simple union of the transducers encoding each individual word-parse pair, as was formulated for finite-state automata by Jurafsky and Martin (2009).

Although the minimization algorithm proposed by Revuz influenced the design of JCLexT's transducer size-reduction algorithm, differences do exist. Firstly, as Revuz formulated his algorithm in a relatively high level pseudo-code, a direct port to Java is difficult due to programming constraints. Secondly, there is a fundamental difference between acyclic DFSAs and FSTs, since the latter are not always determinizable and minimizable, see e.g. Jurafsky and Martin (2009), Beesley and Karttunen (2003), and Allauzen and Mohri (2003). Only p-subsequentializable transducers can be determinized and minimized. By contrast, simple automata, i.e. acceptors, are always determinizable and minimizable.

One important aspect of full-form lexicons is that they typically contain ambiguities and word-parse pairs of unequal length. As a consequence, FSTs compiled from such lexicons with Foma and XFST have arcs labeled with epsilons on the input side and are non-sequential. They are of the type which are classified as restricted by Alencar et al. (2014), being processed much faster than unrestricted FSTs.

Three goals were pursued in designing the compiling algorithm. Firstly, the generated FSTs should be restricted. Secondly, significant compression of the source should be achieved and finally, the FSTs generated should be superior to the FSTs produced by Foma.

3. Test Data

We tested JCLexT with LEXPT01, our own Portuguese lexical transducer with approximately 4 million paths. It was created primarily from FST03, an existing resource containing about 1.3 Million paths (Alencar et al., 2014), by extending its coverage to handle different orthographies and productive word-formation processes.

Foma's `print lower-words` command was applied to LEXPT01, extracting its lower language. Then Foma's `flookup` utility was used to parse this language with LEXPT01. This output was the full-form lexicon DIC.

In order to assess if FSTs generated by JCLexT behaved in an identical manner to analogous FSTs compiled by Foma with respect to unknown words, two corpora were used. The first one, MACM, is a slightly modified version of the Mac-Morpho corpus distributed with NLTK (Bird, Klein and Loper, 2009). The second being a word list from *Projecto Natura* referred to in this work as NAT.[1] Both corpora contain about one million items.

[1] URL: http://natura.di.uminho.pt/download/TGZ/Dictionaries/wordlists/LATEST/wordlist-big-latest.txt.xz, last modification timestamp 2015-4-18 19:37.

4. Evaluation

In order to carry out the evaluation, two FSTs were compiled from the full-form lexicon DIC. These were LEXPT01-J and LEXPT01-F, the J and F referring to the fact they were compiled by JCLexT and Foma. Five aspects were considered in the tests: (i) correctness, (ii) size, (iii) complexity, (iv) compilation times, and (v) parsing times.

Correctness was tested using Foma. The word-parse pairs from LEXPT01-J were thus shown to be isomorphic to DIC. Additionally, both LEXPT01-J and LEXPT01-F were applied to MACM and NAT, the word-parse pairs outputted being identical. This shows that the FST created with JCLexT is valid and correct.

Table 1. Size of the FSTs generated by JCLexT and Foma

	States	Arcs	Paths	Size on disk in bytes
LEXPT01-F	58 331	208 845	3 943 728	3 380 852
LEXPT01-J	58 339	208 029	3 943 728	3 089 547

Table 1 summarizes the results of test (ii), the most important being that JCLexT compiled an FST requiring 8.6% less disk space than the Foma FST.

To assess complexity, two aspects were considered: firstly, if the FST is sequential, meaning it is deterministic on the input side and secondly, if the FST is restricted or unrestricted. To detect if an FST is sequential, it is applied to a list of word forms from the input full-form lexicon using a deterministic parsing algorithm, the output is tested for correctness by comparing it to the input lexicon. To verify the restricted nature of the FSTs, an analogous procedure was applied, using JMorpher's RestrictedFST class. Both LEXPT01-F and LEXPT01-J were found to be non-sequential and restricted. Therefore, by these two criteria, they are of similar complexity.

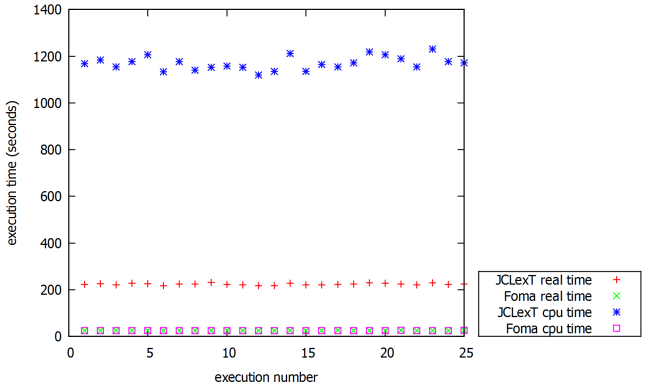

Figure 1: Compiling time of DIC with JCLexT and Foma

Test (iv) and test (v) were carried out on a system with a 64-bit Intel® Core™ Xeon CPU with 8 cores, clocked at 2.10GHz, 32 GB of RAM, running Ubuntu 14.04 LTS. The Linux's `time` utility was used. Figure 1 summarizes the results of test (iv). Foma was about 10 times faster than JCLexT. Counting time utilized on all system processors, JCLexT consumed about 47 times more processor time than Foma.

Test (v) looked at the parsing times of the FSTs generated by both JCLexT and Foma. There was no significant difference in this respect between the two FSTs. JMorpher needed around 5 seconds to parse MACM, either using LEXPT01-F or

LEXPT01-J, while Foma required approximately 4 seconds to parse with either FST.

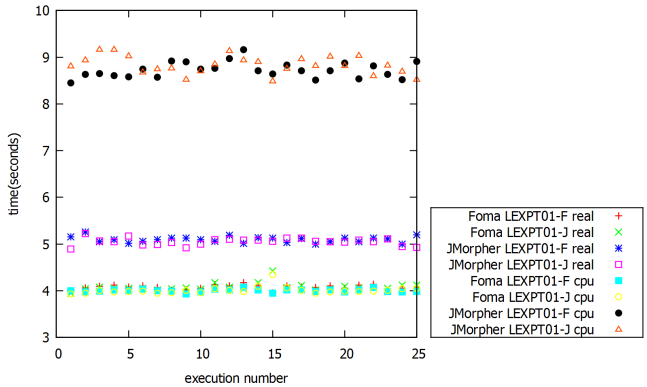

Figure 2: Parsing times of MACM

5. Final Remarks

JCLexT is a tool for compiling FSTs from full-form lexicons, filling the gap left open by Alencar et al. (2014) with the creation of JMorpher, whose Java finite-state tool can only be used for parsing. The ability to generate FSTs from full-form lexicons is very important for resource reuse, since full-form lexicons are available for different languages.

By using a full-form lexicon of Portuguese with approximately 4 million entries, we have compared JCLexT with Foma in relation to transducer equivalence, size and complexity, compiling time and parsing time. JCLexT generated an FST that was equivalent to the Foma counterpart. Both tools produce non-sequential and restricted FSTs, which parse faster than unrestricted FSTs. For the lexicon used in the tests, JCLexT compiled an FST with the same parsing performance as the corresponding FST compiled by Foma, but that is 8.6% smaller.

The decreased size of the FST created by JCLexT means that JCLexT is better suited than Foma to the generation of FST resources for mobile platforms, which often have storage space limitations due to costs. The slower FST production time is negated by the fact that in reality FSTs do not need to be created every time a text needs parsing. Once created a suitable FST could be copied onto many systems to be utilized as needed.

JCLexT combined with the existing JMorpher application means that a complete system using the Java platform to perform the linguistic tasks previously requiring Foma is now available. As a work in progress, the future will hopefully see further compiling algorithm optimization, resulting in faster generation of even smaller FSTs.

One question presented itself to us during this work: Are transducers compiled from large full-form lexicons p-subsequentializable and, thus, determinizable and minimizable? An increase in parsing speed may be possible by investigating if full-form lexicons can be encoded by p-subsequential FSTs and incorporating this feature into our tool.

References

Alencar, Leonel F. de et al. (2014). JMorpher: A Finite-State Morphological Parser in Java for Android. PROPOR 2014, p. 59-69.

Allauzen, C. and Mohri, M. (2003). Finitely subsequential transducers. In *International Journal of Foundations of Computer Science*, 14 (6): 983-994. World Scientific Publishing.

Beesley, K. R. and Karttunen, L. (2003). Finite State Morphology. Stanford, CSLI.

Bird, S., Klein, E., Loper, E. (2009). Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. Sebastopol, CA, O'Reilly.

Hulden, M. (2009). Foma: a Finite-State Compiler and Library. In: Proceedings of the EACL (Demos), p. 29-32.

Jurafsky, D., Martin, J. H. (2009). Speech and Language Processing: an Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. London, Pearson.

Lindén, K., Silfverberg, M., Pirinen, T. (2009) "HFST Tools for Morphology: an Efficient Open-Source Package for Construction of Morphological Analyzers", In: State of the Art in Computational Morphology, Edited by Cerstin Mahlow and Michael Piotrowski, Berlin, Springer, p. 28-47.

Revuz, D. (1992). Minimisation of acyclic deterministic automata in linear time. In *Theoretical Computer Science*, 92 (1): 181-189. Elsevier.

Trommer, J. (2004) "Morphologie", In: Computerlinguistik und Sprachtechnologie: eine Einführung, Edited by Karl-Uwe Carstensen et al., Heidelberg, Spektrum Akademischer Verlag, 2nd edition, p. 190-217.