Finite State Tools for Natural Language Processing

Jan Daciuk
Alfa Informatica, Rijksuniversiteit Groningen
Oude Kijk in 't Jatstraat 26, Postbus 716
9700 AS Groningen, the Netherlands
e-mail: *j.daciuk@let.rug.nl*

May 18, 2000

Abstract

We describe a set of tools using deterministic, acyclic, finite-state automata for natural language processing applications. The core of the tool set consists of two programs constructing finite-state automata (using two different, but related algorithms). Other programs from the set interpret the contents of those automata. Preprocessing scripts and user interfaces complete the set.

1 Introduction

Finite-state automata (both acceptors and transducers) play increasingly important role in natural language processing. Deterministic, acyclic, finite-state automata are used mainly as representations of various kinds of dictionaries. Their main advantages are their small size as compared with the data they hold (see e.g. [3]), and the very fast lookup of strings in an automaton – proportional to the length of the string.

Deterministic, acyclic, finite-state automata (DAFSA) are used in a variety of applications, including DNA sequencing, computer virus detection, and VLSA design. In natural language processing, they are used for spelling correction, restoration of diacritics, morphological analysis, perfect hashing, and acquisition of morphological descriptions for morphological dictionaries. Depending on the application, the contents of an automaton may differ considerably, and so do programs that interpret the automata. However, the basic data structure remains the same. And so do the programs that produce it. It is relatively easy to import data from other systems, as the basic unit in our system is just a string.

2 System Architecture

The architecture of the system is shown on figure 1. The core of the system consists of two programs for construction of DAFSA. They both produce the

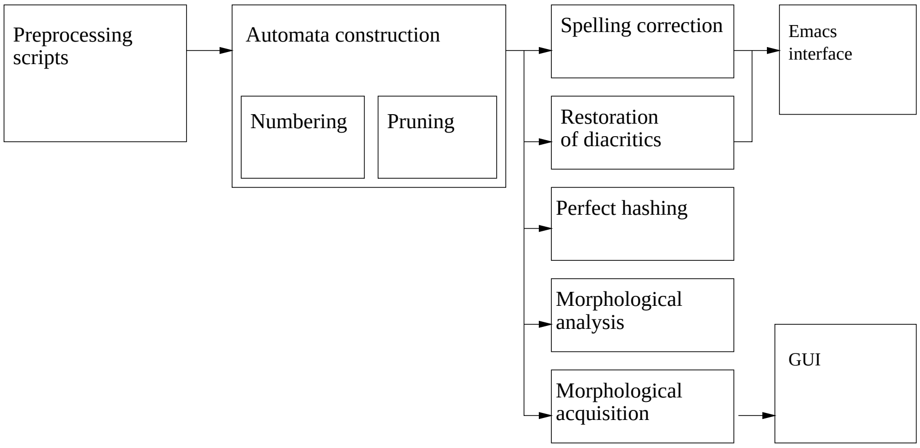

Figure 1: System architecture

same results, but they have different memory requirements and run at different speeds. The algorithms are taken from [2] (newer version has just appeared in [1]). The data for them may be prepared using a variety of preprocessing scripts. The output of the programs is a DAFSA interpreted by other application programs.

Different kinds of applications require different information to be stored in automata. The following sections describe that in detail.

3 Spelling Correction and Restoration of Diacritics

Crude spelling correction requires only a word list. Such a list can easily be obtained from various sources. The system does not provide any scripts for that, as the sources may differ widely, and so the methods of getting the words. However, a conversion from mmorph format to a 3-column format used by tools from the University of Aix-en-Provence is provided. The first column in that format is the inflected word form. Mmorph is Multext morphology tool from ISSCO, Geneva.

The spelling correction tool uses an algorithm by Kemal Oflazer ([7], [6]). Restoration of diacritics is implemented as a simple search with relaxed comparison. An emacs 19 interface can be used to correct words from within that editor.

4 Perfect Hashing

Perfect hashing (see [5], [8]) also requires a list of words. However, the words in the automaton must be numbered. This is done by a special module in the programs that construct automata.

5 Morphological Analysis

Two kinds of morphological analysis are possible using the tool set. The first one is lexicon-based. The outcome is the canonical form, and the categories (or features), or one of them. The strings stored in the automaton consist of two parts. One is the inflected word form to be analyzed, the other - the outcome of the analysis. They are separated with a special character – an annotation separator. This can be seen as an implementation of a p-subsequential transducer. The outcome of the analysis must be coded (see e.g. [4]), because otherwise the automaton would grow to enormous size. Basically, the coding is used to avoid storing the stem more than once in the same string. The morphological analysis program searches for the inflected form in the automaton, and then decodes and outputs the annotation, i.e. the outcome of the analysis. To help in constructing the automaton, several scripts are provided.

It is also possible to analyze words not present in the dictionary. This is done by analyzing the endings, and sometimes the prefixes and infixes (e.g. in case of German). An automaton for approximate morphological analysis (a guessing automaton) associates endings, and sometimes prefixes and infixes as well, with appropriate outcomes of the analysis. But first, those associations need to be created. The system contains several scripts to aid in that process. They invert the inflected form, look for endings, prefixes and infixes, and code them appropriately. The association between an ending and the corresponding analysis is created by inverting the inflected form and appending the analysis as an annotation (similar to the lexicon-based analysis). If prefixes and infixes are present, they are moved from the inflected form to annotations. The resulting strings are data for a guessing automaton. Automata are created in the usual way, and then a specialized module in automata creation programs prunes the structure. If from a given state all paths lead to the same set of annotations, then all states between that state and the annotations can be removed with all their transitions. This significantly reduces the size of the automaton. Further heuristics can be used to improve either recall or precision of the predictions made with such tool. During the analysis, the analyzed word is inverted, and the consecutive letters are looked up in the automaton. When no more letters can be recognized, all annotations reachable from the state the recognition process stopped are decoded as the result of the analysis.

6 Acquisition of Data for Morphological Dictionary

Morphological dictionaries are usually constructed using morphology tools, e.g. two-level morphology. In most advanced tools, a lexeme description is a line containing the base form, categories (or features) including the flectional paradigm, and often the canonical form. It is possible to associate endings, prefixes and infixes with that sort of information in a similar manner to that used in approximate morphological analysis. So the same program that does the approximate morphological analysis is also used (with an appropriate option) for guessing the morphological description of an inflected form. A user runs the program on a list of new words, and the results can be processed using a graphical user interface, where the user can select descriptions, compare them, and see what they produce.

7 Conclusions

We have presented a set of tools based on a simple observation, that DAFSA can be useful in variety of natural language applications. The main data type is an automaton representing a set of strings. For the automata construction programs the strings are just sequences of symbols or characters. This makes it easy to use data from other tools. The meaning is attributed to the strings by application programs that interpret them.

The tools are available in the internet and can freely be used for research purposes. They can handle large data, e.g. they have been used to build a morphological dictionary of German with 3,977,448 inflected forms. It took 20 minutes on a pentium 350MHz computer. They are also very fast. For example, morphological analysis using the same German dictionary is 7.5 times faster than that done by mmorph.

References

[1] Jan Daciuk, Stoyan Mihov, Bruce Watson, and Richard Watson. Incremental construction of minimal acyclic finite state automata. *Computational Linguistics*, 26(1):3–16, April 2000.

[2] Jan Daciuk, Richard E. Watson, and Bruce W. Watson. Incremental construction of acyclic finite-state automata and transducers. In *Finite State Methods in Natural Language Processing*, Bilkent University, Ankara, Turkey, June – July 1998.

[3] Tomasz Kowaltowski, Cláudio L. Lucchesi, and Jorge Stolfi. Minimization of binary automata. In *First South American String Processing Workshop*, Belo Horizonte, Brasil, 1993.

[4] Tomasz Kowaltowski, Cláudio L. Lucchesi, and Jorge Stolfi. Finite automata and efficient lexicon implementation. Technical Report IC-98-02, January 1998.

[5] Claudio Lucchiesi and Tomasz Kowaltowski. Applications of finite automata representing large vocabularies. *Software Practice and Experience*, 23(1):15–30, Jan. 1993.

[6] Kemal Oflazer. Error-tolerant finite state recognition with applications to morphological analysis and spelling correction. *Computational Linguistics*, 22(1):73–89, March 1996.

[7] Kemal Oflazer and Cemalettin Güzey. Spelling correction in agglutinative languages. In *4th Conference on Applied Natural Language Processing*, pages 194–195, Stuttgart, Germany, October 1994.

[8] Emmanuel Roche. Finite-state tools for language processing. In *ACL'95*. Association for Computational Linguistics, 1995. Tutorial.