INTERACTING CROATIAN NERC SYSTEM AND INTEX/NOOJ ENVIRONMENT

BOŽO BEKAVAC, ŽELJKO AGIĆ, MARKO TADIĆ

In this contribution, we present design and implementation details of an early version of Croatian finite state transducer engine called NercFst. The engine currently implements a small subset of Intex/NooJ finite state transducer functionality developed for the purpose of deriving a standalone module for named entity recognition and classification (NERC) system applicable to Croatian texts, previously created as a module in Intex. We also provide some general notes on the Intex module for Croatian NERC and notes on porting the module from Intex to NooJ. Current NercFst engine functionality overview is given in more detail along with some upcoming export features for NooJ which are currently under development with a purpose of supporting portability to various other open source finite state transducer libraries by exporting systems designed and implemented within Intex or NooJ linguistic development environment.

Introduction

Intex and NooJ are well-known powerful environments for developing rule-based natural language processing systems implementable within finite state transducer paradigm and beyond. Implementing versatile finite state transducer backend that encompasses and unites various layers of linguistic processing, using the same visual design principles, also makes Intex and NooJ excellent all-round platforms on which to run these language processing systems.

However, developing large scale natural language processing systems or information retrieval systems – systems that always require several layers of standalone natural language processing black-box modules such as tokenizers, lemmatizers, morphosyntactic taggers and parsers in order to operate – often has very specific demands on technology. For example,

a system for classifying newspaper articles written in Croatian under a classification schema might use both lemmatizer and named entity detection for Croatian as standalone libraries for feature selection at classifier runtime. For such a system to be implemented as a code library itself, both lemmatizer and named entity recognition systems should be deliverable as code libraries, thus preventing the usage of development environments such as Intex or NooJ. While these subsystems – like the Croatian NERC system – might still be developed in Intex or NooJ in form of sets of local regular grammars (i.e. finite state transducers), the classifier system would require another code library capable of running these sets of local grammars in a manner as similar to Intex or NooJ as possible.

Motivation of our work on NercFst system was to create a small and fast finite state transducer engine with just enough capability to run the Croatian NERC system (Bekavac 2005; Bekavac, Tadić 2007), thus making the Croatian NERC system available as a code library for easy integration within natural language processing systems requiring its assistance. On a larger scale, we envisioned a development scheme for finite state transducer based NLP systems in which Intex or NooJ environment is used repeatedly and iteratively throughout the development process – in design, development and testing phases of projects – and then, if explicitly required or otherwise necessary, ported to another FST-capable and desirably open source library for delivering stand-alone modules. This vision also encouraged author of Intex and NooJ to provide some additional export features for NooJ in order to bring the growing community of Intex and NooJ users closer to a large community of programmers using various open source finite state transducer libraries available on the web and utilized in various natural language processing solutions.

Following sections describe the Croatian NERC module and NercFst engine in more detail. We also give insight on interacting Intex and NooJ with other more famous and versatile open source FST engines. Closing sections deal with prospects of evaluation and several directions for future work and experiments in the area.

NERC module for Croatian

In this part, a core of system for Named Entity Recognition and Classification for Croatian Language named OZANA is briefly described

(Bekavac 2005; Bekavac, Tadić 2007). The system is fully implemented in Intex and it is composed of the module for sentence segmentation, general purpose Croatian lexicon (common words), specialized lists of names and local grammars for automatic recognition of numerical and temporal expressions as transducers. The central part of the system is a set of hand-made regular grammars (rules) for recognition and classification of names in tagged and lemmatized texts. Rules are based on certain strategies like internal and external evidence (McDonald 1996) or recognition on sequences of entities in text which belong to same class (Mikheev 1999). Rules are applied using longest match in cascade in defined order (Abney 1996). The results of processing are annotated named entities compliant with MUC-7 specification.

Since Croatian is a highly inflective language, lists of proper names (including all possible generated word-forms) are much bigger in comparison with English or German. For names of persons only, we were using a list of 15,000 male and female personal names accompanied by 56,000 family names registered in the Republic of Croatia (Boras, Mikelić, Lauc 2003:224). This list, expanded to an inflectional lexicon i.e. full word-form list, contains 967,272 lexical entries. Such lists are generated for single and compound forms as well. This implies that system for processing Croatian should be very robust to cope with such amount of lexical entries.

F-measure of the system calculated on informative (newspaper) texts from January 2005 is 0.9. The same rules applied to another genre (textbooks) show a significant drop in the accuracy of the system. Precision is still at 0.79 but recall is at 0.47 thus resulting with F-measure at 0.59. Compared to a similar system for NERC in French texts (Poibeau, Kosseim 2001:148), where also Intex was used as a development environment, we got similar results. System developed for French yielded 0.9 for informative texts and 0.5 in non-informative texts (prose).

NercFst engine for Croatian

NercFst engine – as a finite state transducer library for purposes of extending Croatian NERC system availability – is developed in standard C++ programming language using Microsoft VC 9 compiler and heavily utilizing STL containers and algorithms. It is implemented as set of classes modeling features of finite state transducers (input and output symbols, transducer states, input and output tapes, transitions, configurations and

transition tables) and specifics of Croatian NERC system – among the others: general purpose Croatian lexicon (common words), other specialized lexicons of various named entities (both single- and multi word) and morphosyntactic tag querying. The system is envisioned as code library capable of providing other programmers with an interface of type given in Figure 1.

```
// C++ code illustration
// ...
NercFst nfst;
nfst.initFromFile( exportFile );
nfst.run( inputFile );
nfst.output( outputFile );
// ...
```

Figure 1. NercFst usage illustration

This simple figure clearly illustrates user requirements and system requirements NercFst had to conform to:

1. A single `NercFst` transducer `nfst` is instantiated in computer memory by reading an `exportFile` given in Intex or NooJ format. This implements portability of systems implemented in Intex or NooJ to our NercFst engine. Method `initFromFile()` currently reads Intex files only, namely Intex exports of local regular grammars provided via exporting Intex GRF files to C files by an option of compiling deterministic C transition tables of grammars, containing lists of symbols, states and FST transitions. We chose to implement this option first as Intex determinizes and minimizes transition tables while exporting, thus saving us the effort of implementing FST minimization and determinization algorithms in NercFst. Furthermore, this Intex feature merges all local grammars for a given graph file into a single – minimal and determinized – finite state transducer. Basically, if local grammar LG1 in LG1.grf summons local grammar LG2 defined in LG2.grf, exporting LG1 via Intex creates a single FST containing both LG1 and LG2 symbols and transitions. This saves us the effort of tracking local grammar dependencies as well.
   NooJ finite state transducer engine, as opposed to the one in Intex, performs lazy evaluation of grammars/transducers – transducers are basically compiled at runtime, if required by input data. Therefore,

NooJ does not support exporting nested grammars into single, unique, minimized and deterministic transducers and as a consequence, our NercFst engine is yet to support interaction with NooJ. However, in the following chapter, we discuss work in progress on some new NooJ features that make interaction between its engine and third party FST libraries much easier.

2. A single `NercFst` transducer `nfst` has to be able to process files containing texts written in Croatian. Texts must be tokenized according to Intex and NooJ tokenization principles (Silberztein 2008a; 2008b) before being fed to the transducer or by the `run()` method itself. The method must also be aware of other NERC annotations applied beforehand by other `NercFst` transducer instances, respecting the longest match rule; if for example a DetectInstitutions grammar would detect an institution containing a famous person's name, a DetectPerson grammar applied afterwards should not consider this institution for annotation at all. Also, Croatian NERC utilizes lexicon and morphosyntactic tag lookup. Therefore, `run()` method also had to implement these features. Only single word unit lexicons – general and NE-specific – are currently supported in NercFst via Text Mining Tools library (Šilić et al. 2007) using Intex-specific `<lemma>` syntax. Morphosyntactic tag features and lookup in form `<PoS:subPoS>` is provided by CroTag (Agić et al. 2008) – a morphosyntactic trigram tagger for Croatian. The MULTEXT-East v3 (Erjavec 2004) morphosyntactic tag specification for Croatian was also slightly altered to modify Intex/NooJ requirements and specific NERC system requirements, such as assigning unused specification codes to encode additional features of named entities, e.g. whether lexicon entry is personal name, surname or location instead of just proper noun (coded as `Np` in the standard).

3. A single `NercFst` transducer `nfst` has to provide an option of exporting its output to XML or other format in which NERC system annotation is visible and readable by humans. NercFst engine output is implemented to provide stand-off annotation (we write named entity begin index, end index and tag type to the output), thus exporting it to any other format is not a difficult task. Currently, we support MUC-7 specification (Chinchor 1997) and its tag format. As various grammars could be run sequentially by a single NercFst instance or by a sequence of NercFst instances, additional mechanism also had to be

implemented for merging outputs with respect to running sequences and longest match rules.

4. As NercFst is made to run Croatian NERC system and this system utilized various special keywords provided by Intex and NooJ – these keywords in fact representing specialized local grammars operating inside single tokens, such as <PRE>, <MOT>, <MAJ>, <NB>, etc. – these also had to be supported by NercFst engine. However, being that basic setting of NERC system and NercFst is that symbol equals token and token equals a string as underlying data structure, we had to bypass this token-string rule in order to implement special local grammars as they have a vital role throughout the entire NERC system operation.

As clearly indicated in this specification, NercFst is still in an early development stage and lots of work must be put into improving its features even in order to compare it to other, more mature and more famous code projects implementing finite state transducer functionality. However, if considered solely in context of running Croatian NERC system, it already serves its purpose as it is, although directing its development towards interacting with NooJ instead of Intex, still is on the top of our priority list. Interacting NercFst and other finite state transducer engines with NooJ is discussed in the following chapter.

Interacting NooJ and transducer libraries

As mentioned earlier, Intex combines dependent local grammars into a single finite state transducer structure and makes it both deterministic and minimal. It is available via Intex in internal FST format and C-file format containing three arrays – list of input and output symbols combined, list of states indicating final states and list of transitions containing tuples of integers indicating current state, input symbol and target state of the transducer. With such a feature set, virtually any finite state transducer library could support running Intex grammars, providing that several requirements are supported internally by these libraries. Some of these requirements are presented in the previous chapter and met to some extent by NercFst: (1) supporting specialized lexicon lookup, (2) supporting morphosyntactic tag lookup and (3) implementing keyword-triggered lexical level local grammars. There are surely other features that are supported in Intex and NooJ and not utilized in the Croatian NERC system

so we do not try to make this list comprehensive and exact but, in the contrast, small but indicative.

NooJ environment, as opposed to Intex, evaluates and compiles local grammars in a manner called lazy evaluation – finite state transducers are compiled only when required by input text specifics. This feature – even though it is obviously extremely useful in terms of processing speed when using NooJ as both development and running environment – makes implementing single transition table export feature a more difficult task. Therefore, exporting deterministic transition table is not supported by NooJ, denying NercFst and potentially other FST libraries of intriguing prospects of running transducer systems developed in NooJ.

Guided by these underlying ideas of library interaction, upcoming versions of NooJ environment should – by coordinated work of NooJ developers and NercFst team – at least enable exporting single local grammars from graph files into deterministic transition tables in the AT&T FSM file format (Mohri et al. 1998). This format was chosen because it is directly supported by almost every relevant open source finite state transducer library. Namely, OpenFst (Allauzen et al. 2007) and Helsinki FST toolkit (Koskenniemi, Yli-Jyrä 2008) support it out of the box, while it can be converted with some effort to Stuttgart FST toolbox (Schmid 2005) syntax. In order to fully recreate old Intex functionality with this planned NooJ export feature, single local grammars exported to finite state transducers should be combined by a third-party combining procedure, creating sequences of embedded FSTs and subsequently merging them into a single, minimal and deterministic transducer. Being that all stated finite state transducer libraries implement all the basic transducer operations – among these, union, minimization and determinization are required – the task should not pose difficulties. However, as the export feature is currently under construction by the NooJ development team, it remains to be tested in the future.

Conclusions and future work

We have presented the work-in-progress on NercFst, a FST engine that currently implements a small subset of Intex/NooJ FST functionality developed for the purpose of deriving a standalone module for named entity recognition and classification in Croatian texts. The advantages of described system are: (1) possibility of integration and combination with other natural language processing systems via code library module; (2) possibility of easily redesigning the system to support other systems for processing Croatian language developed in Intex or NooJ; (3) easier

control of processing steps since – from the rule developer's point of view – possibility to approach any data at certain stage of processing could be very useful. For example, during NERC processing some lexical knowledge is relevant only for certain type of discourse. Creation of such resource for temporary lexical data storage is much easier to implement in described system.

Future work directions are most likely to be spread in two possible directions: (1) expanding the NercFst library in order to support additional Intex and NooJ functionality and (2) putting additional effort to interaction of NooJ with open source finite state transducers in general via standard finite state transducer file formats.

Acknowledgement

This work has been supported by the Ministry of Science, Education and Sports, Republic of Croatia, under the grants No. 130-1300646-0645, 130-1300646-1002, 130-1300646-1776 and 036-1300646-1986.

References

Abney, Steven. 1996. Partial Parsing via Finite-State Cascades, Journal of Natural Language Engineering 2 (4): 337–344.

Agić, Željko; Tadić, Marko; Dovedan, Zdravko. 2008. Combining Part-of-Speech Tagger and Inflectional Lexicon for Croatian. Proceedings of the Sixth Language Technologies Conference. Institut Jožef Stefan, Ljubljana, Slovenia, 2008: 116-121.

Allauzen, Cyril; Riley, Michael; Schalkwyk, Johan; Skut, Wojciech; Mehryar Mohri. 2007. OpenFst: A General and Efficient Weighted Finite-State Transducer Library. Proceedings of the Ninth International Conference on Implementation and Application of Automata (CIAA 2007), Springer: 11-23.

Bekavac, Božo; Tadić, Marko. 2007. Implementation of Croatian NERC System, in: Piškorski, Jakub; Tanev, Hristo; Pouliquen, Bruno; Steinberger, Ralf (ed.) Proceedings of the Workshop on Balto-Slavonic Natural Language Processing 2007, ACL, Prague 2007: 11-18.

Bekavac, Božo. 2005. Strojno prepoznavanje naziva u suvremenim hrvatskim tekstovima, PhD dissertation, Faculty of Humanities and Social Sciences, University of Zagreb.

Boras, Damir; Mikelić, Nives; Lauc, Davor. 2003. Leksička flektivna baza podataka hrvatskih imena i prezimena, Modeli znanja i obrada prirodnog jezika, Radovi Zavoda za informacijske studije (vol. 12): 219–237.

Erjavec, Tomaž. 2004. Multext-East Version 3: Multilingual Morphosyntactic Specifications, Lexicons and Corpora. In Proceedings of the Fourth International Conference on Language Resources and Evaluation. ELRA, Lisbon-Paris 2004: 1535-1538.

Friburger, Nathalie; Maurel, Denis. 2004. Finite-state transducer cascades to extract named entities in texts, Theoretical Computer Science, 313(1): 93–104.

Gale, William; Church, Kenneth; Yarowsky, David. 1992. One Sense per Discourse. Proceedings of the 4th DARPA Speech and Natural Language Workshop, Harriman, NY: 233–237.

Gross, Maurice. 1993. Local grammars and their representation by finite automata, Data Description, Discourse. in: Hoey, M. (ed.) Harper-Collins, London: 26–38.

Koskenniemi, Kimmo; Yli-Jyrä, Anssi. 2008. CLARIN and Free Open Source Finite-State Tools. In the Proceedings of FSMNLP 2008, Ispra, Italy.

McDonald, David. 1996. Internal and external evidence in the identification and semantic categorization of proper names. in: Boguraev, I.; Pustejovsky, J. (eds.), Corpus Processing for Lexical Acquisition, MIT Press, Cambridge, MA: 21–39.

Mikheev, Andrei; Grover, Claire; Moens, Marc. 1999. Named Entity Recognition without Gazetteers, Proceedings of the Ninth Conference of the European Chapter of the Association for Computational Linguistics, Bergen: 1–8.

Mohri, Mehryar; Pereira, Fernando; Riley, Michael. 1998. A Rational Design for a Weighted Finite-State Transducer Library. Lecture Notes in Computer Science, 1436, 1998.

Piskorski, Jakub; Neumann, Günter. 2000. An Intelligent Text Extraction and Navigation System, Proceedings of the 6th International Conference on Computer-Assisted Information Retrieval (RIAO'00), Paris.

Poibeau, Thierry. 2000. A Corpus-based Approach to Information Extraction, Journal of Applied Systems Studies, 1(2): 254–267.

Poibeau, Thierry; Kosseim, Leila. 2001. Proper Name Extraction from Non-Journalistic Texts, in: Daelemans, W.; Sima'an, K.; Veenstra, J.; Zavrel, J. (eds.), Computational Linguistics in the Netherlands 2000:

Selected Papers from the Eleventh CLIN Meeting, Rodopi, Amsterdam: 144–157.

Schmid, Helmut. 2005. A Programming Language for Finite State Transducers. Proceedings of the 5th International Workshop on Finite State Methods in Natural Language Processing (FSMNLP 2005), Helsinki, Finland.

Silberztein, Max. 2008. Intex manual, NooJ manual. Available at URLs http://www.nyu.edu/pages/linguistics/intex/ and http://www.nooj4nlp.net/ (2008-12-07).

Šilić, Artur; Šarić, Frane; Dalbelo Bašić, Bojana; Šnajder, Jan. 2007. TMT: Object-Oriented Text Classification Library. Proceedings of the 29th International Conference on Information Technology Interfaces. SRCE, Zagreb, 2007: 559-566.