Open source multi-platform NooJ for NLP

Max SILBERZTEIN[1] *Tamás* VÁRADI[2] *Marko* TADIĆ[3]
(1) UNIVERSITÉ DE FRANCHE-COMTÉ, Besançon, France
(2) RESEARCH INSTITUTE FOR LINGUISTICS MTA, Budapest, Hungary
(3) FACULTY OF HUMANITIES AND SOCIAL SCIENCES ZAGREB UNIVERSITY, Zagreb, Croatia
max.silberztein@univ-fcomte.fr, varadi.tamas@nytud.mta.hu,
marko.tadic@ffzg.hr

ABSTRACT

The purpose of this demo is to introduce the linguistic development tool NooJ. The tool has been in development for a number of years and it has a solid community of computational linguists developing grammars in two dozen languages ranging from Arabic to Vietnamese[1]. Despite its manifest capabilities and reputation, its appeal within the wider HLT community was limited by the fact that it was confined to the .NET framework and it was not open source. However, under the auspices of the CESAR project it has recently been turned open source and a JAVA and a MONO version have been produced. In our view this significant development justifies a concise but thorough description of the system, demonstrating its potential for deployment in a wide variety of settings and purposes. The paper describes the history, the architecture, main functionalities and potential of the system for teaching, research and application development.

KEYWORDS : NooJ, Finite-State NLP, Local Grammars, Linguistic Development Tools, INTEX, Information extraction, text mining

[1] See http://nooj4nlp.net/pages/resources

Proceedings of COLING 2012: Demonstration Papers, pages 401–408,
COLING 2012, Mumbai, December 2012.

1 Introduction

The purpose of the paper and the demonstration is to introduce the NLP system NooJ in its new incarnation as an open-source, cross-platform system. The porting into JAVA, recently created under the supervision of Max Silberztein, the developer of the system, within the CESAR project, as well as the recent steep development path of the core system, justifies a brief overview and demonstration of the potential of the system for a wide variety of computational linguistic applications. For lack of space we will concentrate on two issues: how NooJ is capable of supporting development of a wide variety of grammars of different strength and formalism and how efficiently it can do so. Accordingly, ***Section two*** gives a brief general overview, ***section three*** discusses the architecture, ***section four*** describes the expressive power and suitability of NooJ to develop grammars of various formalisms, ***section five*** will contain a brief discussion of the processing efficiency of the system.

2 General description of NooJ

NooJ is a self-contained corpus analysis and comprehensive linguistic development tool, employing an efficient uniform formalism that enables the system to be deployed for a range of NLP tasks. A more flexible and powerful successor to INTEX (Silberztein, 1993), which was created at the LADL to develop sophisticated yet robust computational linguistic analyses, NooJ has far surpassed its predecessor both in terms of implementation and linguistic and computational power and efficiency (see sections 4 and 5 for details).

3 Architecture

The system consists of three modules, corpus handling, lexicon and grammar development that are integrated into a single intuitive graphical user interface (command line operation is also available). An essential feature of NooJ is that these modules are seamlessly integrated and are internally implemented in finite state technology (with optional important enhancements, see below).

3.1 Corpus handling

NooJ is geared towards developing grammars processing large amounts of texts and therefore contains a full-fledged ***corpus processing module***, indexing, annotation and querying of corpora is a basic functionality. Corpora's size are typically up to 200MB+ (e.g. one year of the newspaper Le Monde), and we have experimented with the MEDLINE corpus (1GB+) successfully. Text can be imported in a wide variety of formats and a lexical analysis is immediately applied on the basis of a robust dictionary module that has a built-in morphological analyser. The result of the lexical analysis becomes the initial tier in a set of stand-off annotation levels containing at first POS and morphological codes as well as the result of any morphological grammars that carried out typical pre-processing normalisations of the text. This basic annotation, amounting to indexing of the corpus, can be enhanced with an arbitrary number of further tiers as subsequent syntactic grammars are applied in a cascaded manner. The corpus handling

facility provides instant feedback for the coverage of grammars in the form of concordances, the annotation of which can be manually filtered before applying to the text. NooJ can handle pre-annotated corpora, providing that the XML annotations that represent lexical information follow a few requirements.

3.2 Lexical module

NooJ contains a robust dictionary module, which, together with its integrated morphological grammars, is designed to handle simple words, affixes and multi-word units. The lexical module is capable of handling full-fledged dictionaries, for example, the Hungarian system consists of cc. 72000 lemmas representing over120 million word forms, which the system can efficiently analyse and generate. Dictionary entries consist of POS and any number of typed features describing the morphological, syntactic, semantic etc. characteristics of the lemma, including, for example, foreign language equivalents.

Morphology is implemented by specifying paradigms that define all possible affix sequences, employing morphological operators to take care of assimilation, stem changes etc. Morphological grammars can be written in a graphical interface to segment and analyse compound word forms, to treat spelling variants as well as to implement guessers to handle unknown words.

3.3 Syntactic module

One of the most attractive features of the system that immediately appeals to users is the ease with which sophisticated grammars (of various levels of computing power) can be built in graph form and applied to corpora as a query (Beesley & Karttunen, 2003)(see, Figure 1, for an example). This feature alone makes NooJ ideal for teaching and rapid application development tool alike. The graphs have alternative textual notation in case that mode of definition proves more applicable.

The original philosophy behind developing NooJ was to employ fast finite state technology to exploit its potential in describing local dependencies in language in all their complex details. This led to the development of local grammars capturing sophisticated distributional distinctions, which are applied in a cascaded manner to yield, hopefully, a comprehensive parsing of text. (Gross, 1997)

Recently, a number of enhancements have been introduced that significantly increased the expressive power and the elegance of grammars that can be devised in NooJ. Such advanced features include use of variables, lexical constraints, agreement over long distance dependencies and the generation of word forms required by the grammar. As a result, in its current stage of development NooJ allows implementation of grammars that represent various levels of computing power. This will be the focus of attention in Section 4.

3.4 Implementation

Originally available only on the .NET and Mono platforms, the system has recently been ported to JAVA. NooJ's engine and its user interface are decoupled and the corresponding API has been defined to ease the task of porting both to Java. The Java

version of NooJ has been implemented using the Swing GUI components. The resulting JAVA system, currently in final integration testing phase, will be published soon together with source code and documentation. NooJ is capable of processing texts in over 150 file formats (including HTML, MSWORD, PDF, RTF, all version of ASCII and Unicode, etc.), NooJ internal engine processes texts in UTF8.

4 The expressive power of NooJ grammars

Beside being a corpus processor (that can process large texts in real time) as well as a linguistic development tool (used to implement large number of linguistic resources), NooJ offers a unified formalism that can be used to enter grammars of the four type of the Chomsky-Schützenberger hierarchy (Chomsky & Schützenberger, 1963):

- NooJ's Regular Expressions and finite-state graphs are compiled into finite-state automata and transducers. For instance, the graph in Figure 1 recognizes the set of correct sequences of preverbal particles in French.

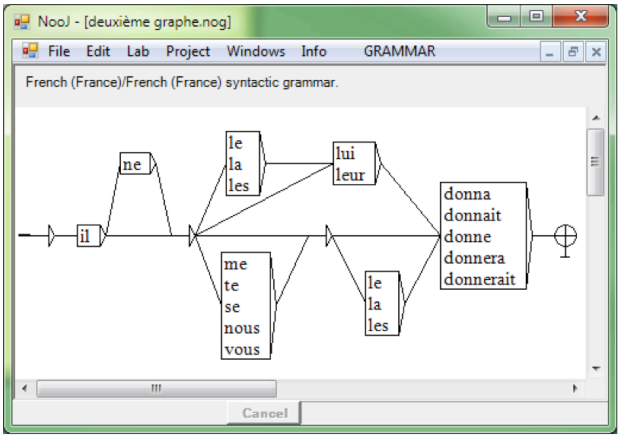

Figure 1 Implementing Type3 grammars

- NooJ's Context-Free Grammars and recursive graphs are compiled into finite-state machines or push-down automata[2]. For instance, the grammar in Figure 2 contains two graphs that recognize transitive sentences in French GN (NounPhrase). <V> (Verb) GN (NounPhrase).
- NooJ's Context-Sensitive grammars have two components: a FS or CFG finite-state component, and a constraint-resolution component. For instance, the graph in

[2] Note that most real world CFGs have either left or right recursions, which can be removed automatically. Left-recursive and right-recursive Context-Free Grammars are turned into equivalent non-recursive grammars and subsequently compiled into finite-state machines.

Figure 3 recognizes the language $\mathbf{a^n b^n c^n}$ in two steps: The finite-state graph recognizes the language a*b*c*, then, for all sequences recognized by the finite-state graph, the constraints: (here: <BLENGTH=ALENGTH> and <CLENGTH=ALENGTH>) are checked.

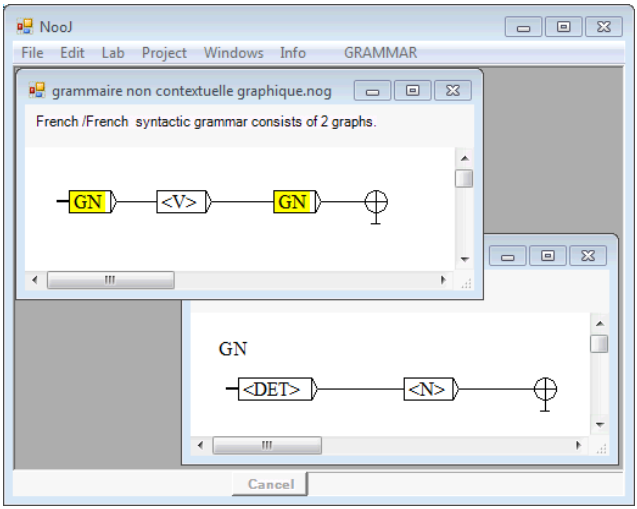

Figure 2 Implementing Context-Free Grammars

- NooJ's transformational grammars also have two components: one finite-state or context-free grammar component, and a component that produces and parses outputs based on variables' values. For instance, the three grammars in Figure 4 compute the Passive form, the negation form and the pronominalized form of a given sentence.

The fact that NooJ's enhanced transducers can produce sequences that can in turn be parsed by the same, or other transducers gives NooJ the power of a Turing-Machine.

In effect, these different types of grammars and machines make NooJ the equivalent to a large gamut of formalisms used in Computational Linguistics, including XFST (Beesley & Karttunen, 2003), GPSG (Gazdar, Klein, Pullum, & Sag, 1985), CCG (Steedman & Baldridge, 2011) and TAG (Joshi & Schabes, 1997), LFG (Kaplan & Bresnan, 1994) and HPSG (Pollard & Sag, 1994). The fact that the same exact notation is used in all four of grammars/machines makes NooJ an ideal tool to parse complex phenomena that involve phenomena across all levels of linguistic phenomena.

5 Efficiency

NooJ's parsers are extremely efficient compared with other NLP parsers. Compared with INTEX, GATE (Hamish Cunningham, 2002) or XFST (which have very efficient finite-

state parsers). NooJ's finite-state graphs are compiled dynamically *during* parsing, instead of being compiled (determinized and minimized) *before* parsing. NooJ often needs to optimize only a fraction of large grammars (typically, 100.000+ states), and ends the parsing of large corpora much faster than it would have been necessary to just fully compile them.

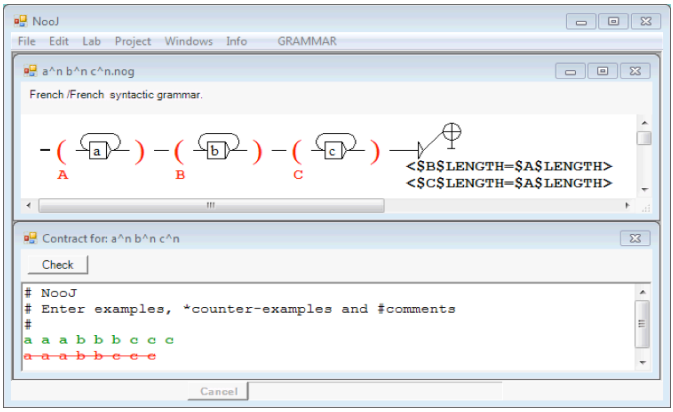

Figure 3 Implementing Context-Sensitive Grammars

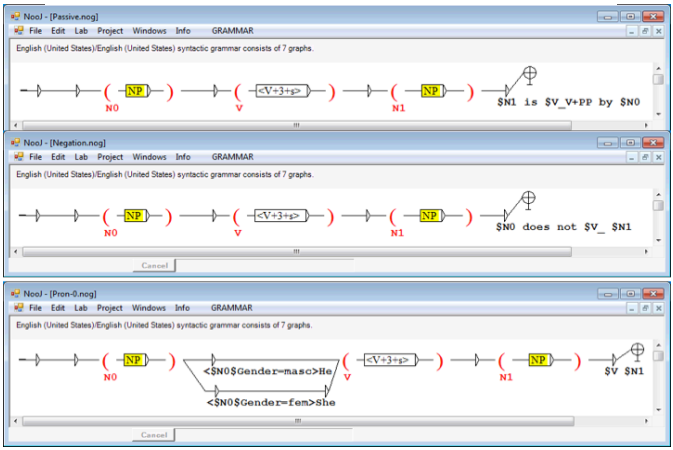

Figure 4 Implementing Unrestricted Grammars

NooJ's parsers are also much more efficient than the parsers that process context-sensitive formalisms such as TAG, CCG or LFG, because when parsing a context-sensitive language, NooJ actually starts by parsing their finite-state superset, and then applies a

series of constraints to filter out the result. See for instance how the context-sensitive language aˆn bˆn cˆn is processed in Figure 3: in a first step, a finite-state automaton recognizes the language a* b* c* in O(n); in the second step, NooJ checks each of the two constraints in constant time. In other words, when applying this context-sensitive grammar (as well as many "classical" grammars used to describe realistic linguistic phenomena), NooJ's parser performs in O(n), as opposed to O(nˆ6) which is the typical efficiency for parsers available for mildly context-sensitive formalisms such as TAG.

6 Resources

There are over 20 language modules already available for download from NooJ's WEB site www.nooj4nlp.net, including 5 that have open source dictionaries and morphology. When the open source of NooJ is published, the goal of the METANET CESAR consortium is to release open source modules for the languages covered by the CESAR project.

NooJ provides half a dozen tools to help linguists develop new language modules rapidly. The first versions of the latest NooJ modules (Turkish and Serbian) have been developed in less than a year.

7 Conclusions

The recent porting to JAVA, the publication of its source code and, more importantly, its enhanced features developed in recent years suggest that the NooJ system has reached a major milestone in its development. We hope that this paper and the live demonstration will serve to argue the case that these major developments open up new perspectives for a wider range of application within the computational linguistic community.

References

Beesley, R. K., & Karttunen, L. (2003). *Finite State Morphology*. Stanford: CSLI Studies in Computational Linguistics.

Bresnan, J. (Ed.). (1982). *The Mental Representation of Grammatical Relations*. Cambrdige, MA: MIT Press.

Chomsky, N., & Schützenberger, M. P. (1963). The algebraic theory of context free languages. In P. Braffort, & D. Hirchberg (Eds.), *Computer Programming and Formal Languages* (pp. 118-161).

Gazdar, G., Klein, E. H., Pullum, G. K., & Sag, I. A. (1985). *Generalized Phrase Structure Grammar*. Oxford: Blackwell.

Gross, M. Lexicon-Grammar. The Representation of Compound Words. *COLING 1986*, (pp. 1-6).

Gross, M. (1997). The Construction of Local Grammars. In E. Roche, & Y. Schabes (Eds.), *Finite State Language Processing* (pp. 329-352). Cambridge, Mass.: MIT Press.

Hamish Cunningham, D. M. (2002). GATE: an Architecture for Development of Robust HLT Applications. *Proceedings of the 40th Anniversarsary Meeting of the ACL* (pp. 168-175). Philadeplphia: ACL.

Joshi, A. K., & Schabes, I. (1997). Tree Adjoining Grammars. In G. Rozenberg, & A. Salomaa, *Handbook of Formal Languages* (Vol. 3, pp. 69-120). Berlin: Springer Verlag.

Kaplan, R. M., & Bresnan, J. (1994). Lexical-Functional Grammar: A Formal System for Grammatical Representation. In M. Dalrymple, R. M. Kaplan, J. T. Maxwell, & A. Zaenen (Eds.), *Formal Issues in Lexical-Functional Grammar* (Lecture Notes No. 47 ed., pp. 29-131). CSLI Publications.

Pollard, C., & Sag, I. A. (1994). *Head-driven phrase structure grammar.* Chicago: Unversity of Chicago Press.

Silberztein, M. (1993). *Dictionnaires électroniques et analyse automatique de textes : le système INTEX.* Paris: Masson Ed.

Silberztein, M. (1998). INTEX: An integrated FST toolbox. In D. Wood, & S. Yu, *Automata Implementation* (pp. 185-197). Berlin, Heidelberg: Springer.

Steedman, M., & Baldridge, J. (2011). Combinatory Categorial Grammar. In R. Borsley, & K. Borjars (Eds.), *Non-Transformational Syntax* Oxford: Blackwell pp. 181-224..