Applications of Finite-State Language Processing

Applications of Finite-State Language Processing: Selected Papers from the 2008 International NooJ Conference

Edited by

Tamás Váradi, Judit Kuti and Max Silberztein

CAMBRIDGE
SCHOLARS

P U B L I S H I N G

Applications of Finite-State Language Processing:
Selected Papers from the 2008 International NooJ Conference,
Edited by Tamás Váradi, Judit Kuti and Max Silberztein
Technical Editors: Iván Mittelholcz, Judit Kuti

This book first published 2010

Cambridge Scholars Publishing

12 Back Chapman Street, Newcastle upon Tyne, NE6 2XX, UK

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

ISBN (10): 1-4438-2573-5, ISBN (13): 978-1-4438-2573-3

TABLE OF CONTENTS

PREFACE OF THE EDITORS

The present volume contains papers from the 2008 International NooJ conference held between 08.06.2008. and 10.06.2008. in Budapest. While the focus of the Budapest conference was on making NooJ compatible with other applications, the papers vary with respect to whether they regard NLP as a research goal or as a tool. However, they all present a slightly different problem either in the field of NLP, or in one that can be solved using NLP, or present a new development in the tool itself.

As for recent technical developments, NooJ's developer, **Max Silberztein** introduces new achievements in the software, focusing this year on extending the system's disambiguation capabilites. The conference's invited speaker, **Kimmo Koskenniemi**, envisages more compatibility between different finite-state NLP tools developed for similar purposes – he outlines the possibility of a FST platform that could act as an interface between different morphological analysers written in different styles. It is in this spirit that **Bekavac and his colleagues** present the early version of a Croatian finite state transducer engine (NercFst), which currently implements a small subset of 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. **Matos and Maia** describe the integration of NooJ into Corpógrafo, a freely accessible web-based system for corpora, terminology, lexicology and phraseology research in Portuguese. Also for Portuguese, **Anabela Barreiro** describes a tool that uses Port4NooJ – a set of linguistic resources developed with NooJ for the automated processing of Portuguese – and other enhanced linguistic resources for the automatic paraphrasing of mainly support verb constructions, as a preparation of texts for MT. Focusing on verbs, as well, **Vučković and her colleagues** present how verb valency data, added to the dictionary, enhances the recognition of VP-, NP- and PP-chunks. **Zoltán Alexin**'s paper reports on constructing a finite state automaton using NooJ for morphological analysis in Hungarian, in which the number of states of the automaton is decreased by the theoretically largest possible percentage. **Kata Gábor** explains how a rule-based shallow-parsed corpus of 10 million words has been created using Hungarian NooJ.

Cristina Mota's paper shows how combining Collins' and Singer's semi-supervised co-training algorithm with NooJ enables the recognition of certain Named Entities. **Chadjipapa and her colleagues** focus on compound and proper noun treatment in the Greek NooJ module, while **Fehri and her colleagues** introduce the reader to the challenges of automatic recognition and semantic analysis of Arabic Named Entities with NooJ. Staying with Arabic, **Slim Mesfar**'s contribution is a response to the challenge of building an automatic morpho-syntactic analyser for the Arabic language – the written form of which poses an extra amount of problems to the researcher, which someone dealing with a European language would never encounter. **Huei Chi-Lin**'s paper leads us through the necessary steps of creating a Chinese language encoded corpus for further use in NooJ – giving insight into several further problems of language processing. **Simonetta Vietri** reports of the formalisation of a Lexicon-Grammar table in a NooJ Dictionary/Grammar pair. **Milos Utvić** describes the expansion of the Serbian NooJ-dictionary by the processing of regularly derived word forms. **Staša Vujičić** and **Duško Vitas** deal with the problem of recognising a special class of Named Entities, namely odonyms (names of streets) using NooJ.

Gucul-Milojević and her colleagues look into the representation of women in Serbian newspaper texts, using NE recognition and local grammars in NooJ. **István Csűry** looks at the problem of identifying constituents of semantic constructions in order to annotate a corpus with information on discourse markers and outlines then some lexical and contextual features pointing towards a solution. **Bea Ehmann** and **Vera Garami** give insight into a joint research in the field of scientific narrative psychology, wherein NooJ provides the language technological background to make inferences about processes time experience based on text corpora. And last but not least, **Helène Pignot** and **Odile Piton** report of the language processing challenges of 17th century English travellers' literature.

As this short overview hopefully shows, the range of problems dealt with in the volume is quite varied, which will hopefully enable the readers to find contributions that are relevant to their field of interest.

10.02.2010. The Editors

Disambiguation Tools for NooJ

Max Silberztein

Introduction

With NooJ, linguists can represent five types of Atomic Linguistic Units (ALUs)[1]:

- •Affixes, morphemes and components of contracted words, e.g. *dis-, -ization*, *cannot*
- •simple words and their morphological variants, e.g. *a laugh, to laugh, laughed, laughable*
- •multi-word units, semi-frozen terms and their variants, e.g. *as a matter of fact, a nuclear submarine*
- •local syntactic units, e.g. complex determiners, dates, e.g. *Most of my groups of*, *Monday June 5th in the early afternoon*
- •discontinuous expressions such as collocations, support verb constructions, phrasal verbs, e.g. *to take ... into account, to give ... in*

Accordingly, NooJ provides tools to describe these five types of ALUs and recognize them in corpora automatically: dictionaries, morphological Finite State Transducers (FSTs), Recursive Transition Networks (RTNs), as well as the ability to link dictionaries and syntactic grammars to formalize lexicon-grammar tables[2].

Each level of analysis constitutes an autonomous module. NooJ processes each module one after the other, in cascade form. Therefore, each module must produce a result that has 100% recall, so that the subsequent module processes an input which lacks no potential linguistic hypothesis. The cost of this approach is that the accuracy of each module is very low: all potential linguistic hypotheses have to be produced and transmitted to the next module, however improbable they might be.

The accuracy of a lexical parser will get lower and lower as the precision of the linguistic data increases: for instance, a simple parser might process the word form *will* as two-time ambiguous (Noun or Verb) whereas a more sophisticated parser will distinguish the different meanings of the noun *will* (a mental faculty, a legal declaration) as well as

[1] See the NooJ manual, which is updated regularly (Silberztein 2003).

[2] See (Vietri 2008) for examples of lexicon-grammar tables and their formalization in NooJ.

of the verb (used to introduce the future, or synonymous to *to wish*). Clearly, the more sophisticated lexical parser produces a higher level of ambiguity than the simple one, and it takes a more sophisticated linguistic analysis to solve the extra ambiguities.

An ideally tagged text

Because any linguistic analysis must process all types of ALUs (not only the simple words), an ideal tagger should produce a result that looks like the following:

> Battle-tested/A Japanese/A industrial managers/N here/ADV always/ADV buck up/V nervous/A newcomers/N with/PREP the/DET tale/N of/PREP the first of their/N countrymen/N to/PREP visit/V Mexico/LOC, a boatload of/DET samurai warriors/N blown ashore/VPP 375 years ago/DATE. From the beginning/DATE, it took/EXP1 a/DET man/N with/PREP extraordinary/A qualities/N to/EXP1 succeed/V in/PREP Mexico/LOC, says/V Kimihide Takimura/NPR, president/N of/PREP Mitsui/NPR group's/N Kensetsu Engineering Inc./ORG unit/N.

For instance, it is crucial to tag the multi-word noun *industrial managers* as a unit, as opposed to produce the two tags "industrial/A" and "managers/N". In most cases, the semantic analysis of the sequence "industrial <N>" is:

Industrial <N> = <N> produced by industrial methods

This productive analysis can successfully be applied to a large number of nouns, such as in the following examples:

Industrial cheese = cheese produced by industrial methods
Industrial food = food produced by industrial methods
Industrial cloth = cloth produced by industrial methods
etc.

But this productive analysis does not apply to *industrial manager*. If one wants to translate this sequence correctly into French, one has to translate it as a whole, and not word by word. For instance, a correct French translation would be "patron de PME", whereas "gestionnaire industriel" sounds odd at the very least, and is never used.

If it was possible to automatically tag texts *correctly*, i.e. to automatically annotate all types of linguistic units (and not only simple words), and to produce a 100% correct result, then the example above would constitute a good way to represent the result of the lexical analysis of texts.

Unfortunately, any lexical parser that aims to represent all types of linguistic units while producing a result with 100% recall will produce a high degree of ambiguities, because it is not always possible to remove all ambiguities at the lexical level. For instance, consider the following text:

... There is a round table in room A32 ...

The only way an automatic parser could (maybe!) reliably choose between the analysis "round table = meeting" and the analysis "round table = round piece of furniture" is to perform some complex discourse analysis that would take a larger context into account. It is impossible to choose between these two solutions at a lexical level, i.e. at the level taggers and lexical parsers operate. Taggers designed to remove ambiguities at any cost, even when it is impossible to do so reliably, simply ignore multi-word units and use probability or other techniques-based heuristics to "flip coins" and therefore produce results that are useless for many precise NLP applications. Ambiguities are generated at each level of the five lexical analyses. For instance:

•the morphological parser analyzes the word form *recollect* as the prefix *re* followed by the verb *collect* (meaning: collect again), whereas the dictionary lookup will analyze it as a simple verb (meaning: to control oneself). It would take a sophisticated semantic analysis (at least) to choose between the two solutions in the following text:

John recollects the old coins

Does he remember about them, or did he decide to take his collecting hobby back up again?

•the multi-word recognizer analyses the sequence *acid rock* as a noun (a style of rock music), whereas the simple word recognizer analyses it as a sequence of an adjective followed by a noun.

•there are several phrasal verb entries for "to back up", depending on the distributional property of their object complements. How could a lexical parser choose between these entries without a distributional analysis of the complement, and without a reference analysis when the complement is implicit? For instance:

John backed his car up → John backed it up (he drove in reverse)
John backed Mary's statement up → John backed it up (he supported her statement)

John backed Mary's idea up → John backed it up (he proved her idea right)
John backed his computer up → John backed it up (he saved its files)

In conclusion: it is impossible to build an automatic lexical parser that would disambiguate all the linguistic units that occur in texts. Automatic taggers, which aim at this very goal, simply cannot be relied upon. The only way to build a *reliable* lexical parser is to allow it to represent unsolvable lexical ambiguities. These ambiguities will have to be passed to subsequent parsers that would use syntactic, semantic and/or discourse analysis techniques to solve them (and not in all cases, because there are ambiguous sentences in texts!).

NooJ's Text Annotation Structure (TAS)

NooJ's lexical parser uses parallel annotations, rather than linear tags, to represent the lexical analyses of texts[3]. Annotations have two advantages over tags:

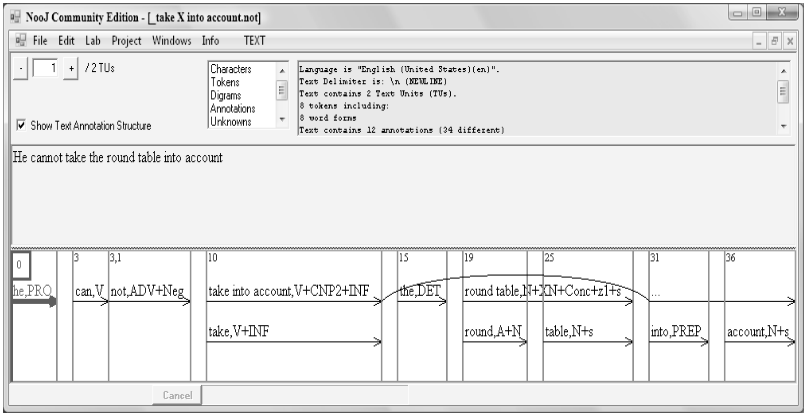
- they can represent all types of linguistic units, such as affixes (inside word forms), multi-word units and discontinuous linguistic units[4];
- they can be stacked together and therefore represent lexical ambiguities.

For instance, consider the following sentence:

He cannot take the round table into account

In this sentence, the word form *cannot* corresponds to a sequence of two linguistic units; the sequence *round table* is ambiguous, and the linguistic unit *take into account* is discontinuous.

[3] See (Silberztein 2006) for a description of NooJ's Annotation engine, and (Silberztein 2007) for linguistic applications of the Text Annotation Structure.
[4] See in particular how discontinuous expressions are represented in the Text Annotation Structure in (Silberztein 2008).


NooJ Community Edition - [_take X into account.not]
File Edit Lab Project Windows Info TEXT
1 / 2 TUs
Characters
Tokens
Digrams
Annotations
Unknowns
Show Text Annotation Structure
0
he,PRO
3
can,V
3,1
not,ADV+Neg
10
take into account,V+CNP2+INF
take,V+INF
15
the,DET
19
round table,N+XN+Conc+z1+s
round,A+N
25
table,N+s
31
...
into,PREP
36
account,N+s
Cancel

```
Language is "English (United States)(en)".
Text Delimiter is: \n (NEWLINE)
Text contains 2 Text Units (TUs).
8 tokens including:
8 word forms
Text contains 12 annotations (34 different)
```

He cannot take the round table into account

Fig. 1. The analysis of the sentence
„He cannot take the round table into account"

The TAS's main application is that no potential linguistic unit is left out: for instance, if the next sentence in the text is: "we will have to postpone it", a semantic parser can infer that round table refers to a meeting, not to a piece of furniture. Conversely, if the next sentence in the text is: "we will have to move it closer to the window", the semantic parser can infer that the round table is a piece of furniture. Both solutions are alive and ready to be chosen.

Another advantage of the TAS is that it unifies all types of linguistic units. From now on, any subsequent parser will process annotations rather than affixes, simple or multi-word units and discontinuous expressions. For instance, the following NooJ query:

`<V> <ADV> <V>`

which stands for: extract from the corpus all sequences of a verb, followed by an adverb, followed by a verb, will produce concordances in which various types of sequences are displayed:

... John has often taken ...

... He cannot take into account the ...

... She is not finished with ...

This characteristic gives NooJ a higher recall factor than other corpus linguistic tools, and is absolutely crucial when dealing with agglutinated languages. For instance, in Arabic, "and in my house" is written as a single word, but NooJ will still process it as a sequence of four distinct annotations. Beyond the simple corpus linguistics applications, the fact that the lexical parser deals with complex word forms allows the subsequent syntactic grammars to be much easier to read and write. For instance, when linguists formalize Romance Languages' noun phrases, they need to pay special attention to the determiner, which is often contracted with the preceding preposition or adverb. For instance, in French, the word form *du* is either a determiner or the contraction of the preposition *de* followed by the determiner *le*. In practice, this simple problem often leads to an unacceptable number of duplicate and redundant rules in the formalization of noun phrases.

With NooJ, however, linguists just have to describe the head of noun phrases independently from the contraction problem, which is solved beforehand by the lexical parser.

Disambiguated texts are useful

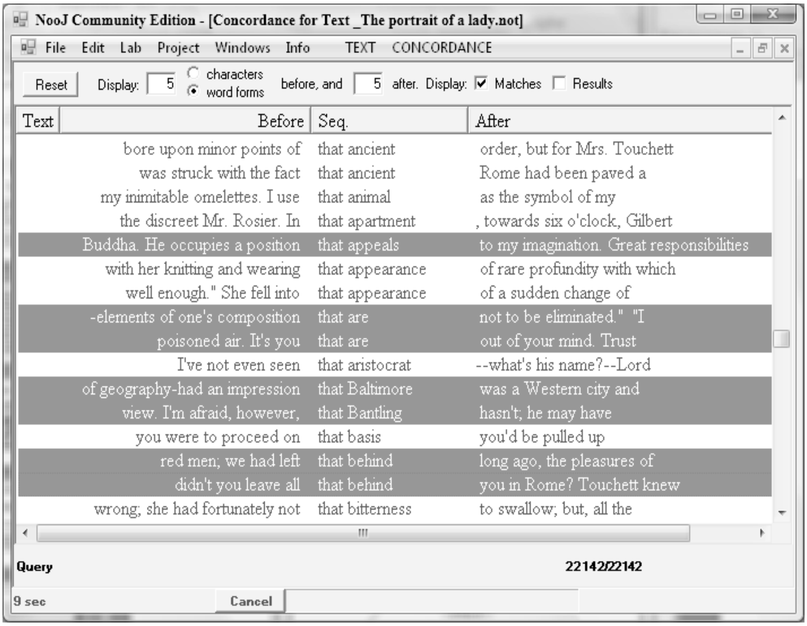
Unfortunately, using fine-grained lexical resources with a TAS in which all ambiguities are retained makes NooJ's corpus processing system produce noisy results that are very unattractive to linguists who are used to tagger-based corpus processors that produce much less noise, at the expense of some silence (but of course users don't see the silence).

A more serious problem is related to the use of NooJ as a corpus processing system, especially by non-linguists who want to analyze texts in other applications: literary, psychological, sociological analyses, etc.

Leaving a large number of ambiguities in TAS often produce very *noisy* results when a user applies a simple query. For instance, the simple query:

```
<DET> <N>
```

(look for all determiners followed by a noun), when applied to the text *The Portrait of a Lady* (Henry James, 1881) produces a 22,142-entry concordance with almost 50% noise:


NooJ Community Edition - [Concordance for Text _The portrait of a lady.not]
File
Edit
Lab
Project
Windows
Info
TEXT
CONCORDANCE
Reset
Cancel

Display: 5 ( ) characters (●) word forms before, and 5 after. Display: [✓ Matches] Results

Text	Before	Seq.	After
	bore upon minor points of	that ancient	order, but for Mrs. Touchett
	was struck with the fact	that ancient	Rome had been paved a
	my inimitable omelettes. I use	that animal	as the symbol of my
	the discreet Mr. Rosier. In	that apartment	, towards six o'clock, Gilbert
	Buddha. He occupies a position	that appeals	to my imagination. Great responsibilities
	with her knitting and wearing	that appearance	of rare profundity with which
	well enough." She fell into	that appearance	of a sudden change of
	-elements of one's composition	that are	not to be eliminated." "I
	poisoned air. It's you	that are	out of your mind. Trust
	I've not even seen	that aristocrat	--what's his name?--Lord
	of geography-had an impression	that Baltimore	was a Western city and
	view. I'm afraid, however,	that Bantling	hasn't; he may have
	you were to proceed on	that basis	you'd be pulled up
	red men; we had left	that behind	long ago, the pleasures of
	didn't you leave all	that behind	you in Rome? Touchett knew
	wrong; she had fortunately not	that bitterness	to swallow; but, all the

Query 22142/22142

9 sec

Fig. 2. The results of the simple query **`<DET> <N>`** when applied to *The Portrait of a Lady* (Henry James, 1881)

(incorrect matches are selected in the figure). This result is arguably improvable! Let's examine the incorrect results:

... that appeals ...

This sequence has wrongly been brought up because *that* could be a determiner, and *appeals* could be a noun. That is not the case though: if *that* were a determiner, we know that it would be a singular one because it is described as such in the English dictionary. However, if the word form *appeals* were a noun, it would be in the plural. The sequence "that/DET appeals/N" violates the agreement rule between determiners and the following noun; therefore this sequence cannot correspond to the query <DET> <N>. We can remove a large number of incorrect results as can be seen if we replace the flawed query <DET> <N> with the more precise one:

`<DET+s> <N+s> + <DET+p> <N+p>`

This query reduces the size of the concordance from 22,142 to 8,517 entries!

... Elements of one's composition that are not to be eliminated...
... It's you that are out of your mind...
... didn't you leave all that behind you in Rome?...

(*are* is a noun that represents a metric unit of area). A syntactic parser could help NooJ decide that in these cases *that* is a pronoun and thus remove these results.

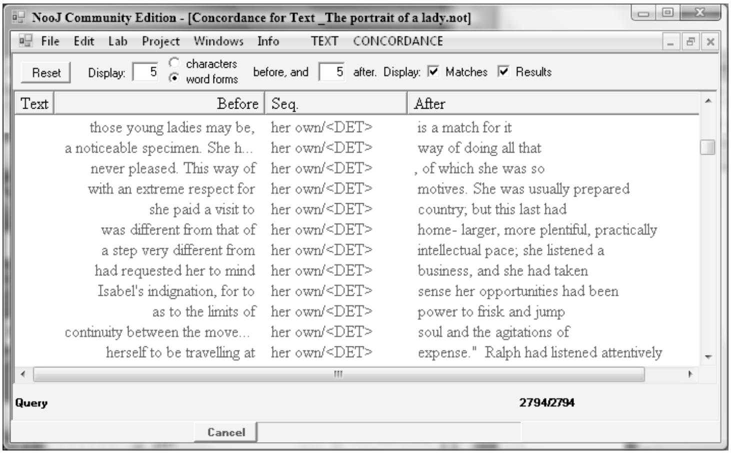
... that Baltimore was a Western city ...
... that Bantling hasn't ...
... we had left that behind long ago ...

However, in these cases, even if NooJ had performed a full syntactic analysis of these contexts, it would still have no way of rejecting these results: from a strictly linguistic point of view, *that* could still be a determiner and *behind* could be a noun.

In conclusion: there are some sequences such as "that appeals" that can be disambiguated with minimum effort: local grammars constitute a very efficient way to remove a large number of ambiguities and thus to clean up the result of users' queries. We do not yet have a full syntactic parser that would help remove a number of ambiguities; however, even if we had a full syntactic parser, there would still be a number of residual ambiguities: it would thus be unavoidable to give users the possibility of removing ambiguities by themselves.

Automatic disambiguation

Local grammars can be used to remove ambiguities automatically. For instance, consider the local grammar in Appendix A that can be used to disambiguate half a dozen very frequent grammatical words which are systematically ambiguous (determiner or pronoun). The grammar merely lists a number of "unambiguous" contexts in which one can disambiguate these words definitively. For instance, *her* followed by *own* has to be a determiner as we can easily see in the concordance below. Although this local grammar is very specific, it covers 2.5% of the text. Our goal is to develop a set of 30+ local grammars such as this one: target frequent ambiguous words to be as efficient as possible, while at the same time as specific as possible, so as to avoid producing incorrect results.


NooJ Community Edition - [Concordance for Text _The portrait of a lady.not]
File Edit Lab Project Windows Info TEXT CONCORDANCE
Reset
Cancel

Display: 5 ( ) characters (•) word forms before, and 5 after. Display: [✓ Matches] [✓ Results]

Text	Before	Seq.	After
	those young ladies may be,	her own/<DET>	is a match for it
	a noticeable specimen. She h...	her own/<DET>	way of doing all that
	never pleased. This way of	her own/<DET>	, of which she was so
	with an extreme respect for	her own/<DET>	motives. She was usually prepared
	she paid a visit to	her own/<DET>	country; but this last had
	was different from that of	her own/<DET>	home- larger, more plentiful, practically
	a step very different from	her own/<DET>	intellectual pace; she listened a
	had requested her to mind	her own/<DET>	business, and she had taken
	Isabel's indignation, for to	her own/<DET>	sense her opportunities had been
	as to the limits of	her own/<DET>	power to frisk and jump
	continuity between the move...	her own/<DET>	soul and the agitations of
	herself to be travelling at	her own/<DET>	expense." Ralph had listened attentively

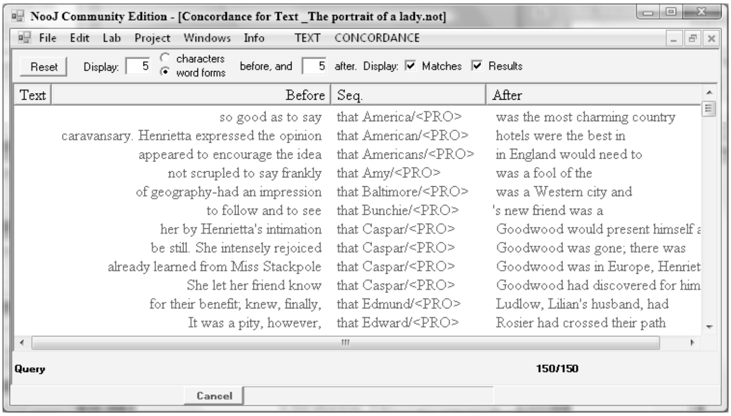
Query 2794/2794

Fig. 3. Concordance for the word *her* in the text *The portrait of a lady*

Considering the context, *her own* is enough to disambiguate *her* as a determiner.

Semi-automatic disambiguation

The double constraint of efficiency and correctness is often very hard to follow, and in a number of cases it is much too tempting to design very efficient rules which are correct in all but a few cases. For instance, it seems that all occurrences of the word *all* followed by a proper name correspond to the pronoun:


NooJ Community Edition - [Concordance for Text _The portrait of a lady.not]
File Edit Lab Project Windows Info TEXT CONCORDANCE
Reset
Cancel

Display: 5 ( ) characters (•) word forms before, and 5 after. Display: [✓ Matches] [✓ Results]

Text	Before	Seq.	After
	so good as to say	that America/<PRO>	was the most charming country
	caravansary. Henrietta expressed the opinion	that American/<PRO>	hotels were the best in
	appeared to encourage the idea	that Americans/<PRO>	in England would need to
	not scrupled to say frankly	that Amy/<PRO>	was a fool of the
	of geography-had an impression	that Baltimore/<PRO>	was a Western city and
	to follow and to see	that Bunchie/<PRO>	's new friend was a
	her by Henrietta's intimation	that Caspar/<PRO>	Goodwood would present himself a
	be still. She intensely rejoiced	that Caspar/<PRO>	Goodwood was gone; there was
	already learned from Miss Stackpole	that Caspar/<PRO>	Goodwood was in Europe, Henriet
	She let her friend know	that Caspar/<PRO>	Goodwood had discovered for him
	for their benefit; knew, finally,	that Edmund/<PRO>	Ludlow, Lilian's husband, had
	It was a pity, however,	that Edward/<PRO>	Rosier had crossed their path

Query 150/150

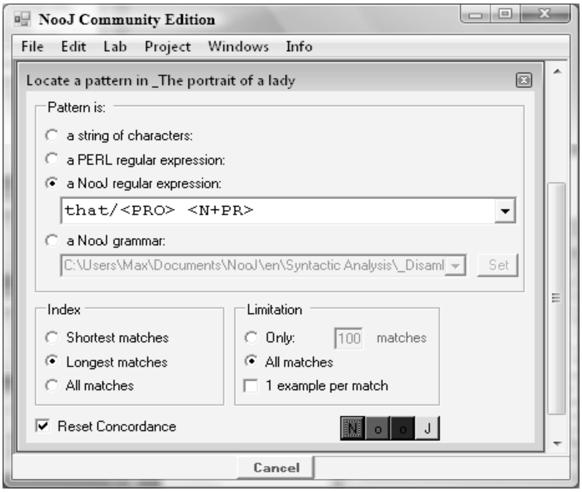
Fig. 4. Concordance for the word *that* in the text *The portrait of a lady*

The context *that <N+PR>* is a good candidate for a disambiguation rule.

This rule would have been very helpful to get rid of the incorrect concordance entries for the above-mentioned query <DET> <N>. Indeed, in the text *The Portrait of a Lady*, this rule can effectively be used to remove 150 ambiguities. However, it is not very difficult to build an example in which *that* is followed by a proper name, but is still a determiner:

Are you speaking of that John Doe?

Therefore, we need to be able to quickly enter simple disambiguation rules and check them before effectively applying them.



Locate a pattern in _The portrait of a lady

Pattern is:
- a string of characters:
- a PERL regular expression:
- [✓ a NooJ regular expression:] `that/<PRO> <N+PR>`
- a NooJ grammar: C:\Users\Max\Documents\NooJ\en\Syntactic Analysis_Disaml Set

Index
- Shortest matches
- [✓ Longest matches]
- All matches

Limitation
- Only: 100 matches
- [✓ All matches]
- ☐ 1 example per match

☑ Reset Concordance

Fig. 5. A query in the form of an expression associated with an output

In NooJ's new v2.2 version, it is possible to type in a query in the form of an expression associated with an output, see Fig. 5.

The regular expression **`that/<PRO> <N+PR>`** matches all the sequences of *that* followed by a proper name; then the output **`<PRO>`** is used to disambiguate the word form *that*.

We can edit the resulting concordance in order to filter out incorrect matches, i.e. sequences that were recognized but should not be used. We first select the unwanted concordance entries, then use the command **Filter out selected lines** of the CONCORDANCE menu to remove them from the concordance, and then Annotate Text (add/remove annotations) to perform the disambiguation for all remaining concordance entries.

This process is very quick: users should not hesitate to enter disambiguation commands that are very efficient but have poor accuracy, such as:

`are/<V>`

`for/<PREP>`

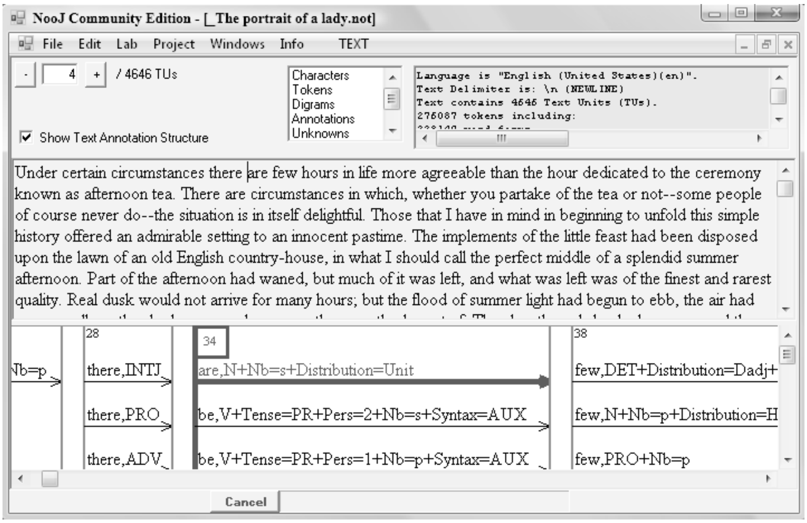
`its/<DET>`

etc.

The possibility of examining the resulting concordances and then filtering out incorrect analyses allows us to experiment more freely. Finally, note that we can save concordances at any moment: this allows us to validate large concordances at our own pace.

Editing the TAS

Sometimes it is much faster to just delete unwanted annotations without having to write local grammars or even queries. In NooJ's new 2.2 version we can just open the TAS (click **Show Text Annotation Structure** at the top of the text window), click an unwanted annotation, and then delete it either with the menu command **Edit > Cut**, or by pressing any of the following keys: Ctrl-x, Delete or Backspace. NooJ manages an unlimited number of undo's: **Edit > Undo** or press Ctrl-Z.



[✓ Show Text Annotation Structure]

```
Language is "English (United States)(en)".
Text Delimiter is: \n (NEWLINE)
Text contains 4646 Text Units (TUs).
276087 tokens including:
```

Under certain circumstances there are few hours in life more agreeable than the hour dedicated to the ceremony known as afternoon tea. There are circumstances in which, whether you partake of the tea or not--some people of course never do--the situation is in itself delightful. Those that I have in mind in beginning to unfold this simple history offered an admirable setting to an innocent pastime. The implements of the little feast had been disposed upon the lawn of an old English country-house, in what I should call the perfect middle of a splendid summer afternoon. Part of the afternoon had waned, but much of it was left, and what was left was of the finest and rarest quality. Real dusk would not arrive for many hours; but the flood of summer light had begun to ebb, the air had

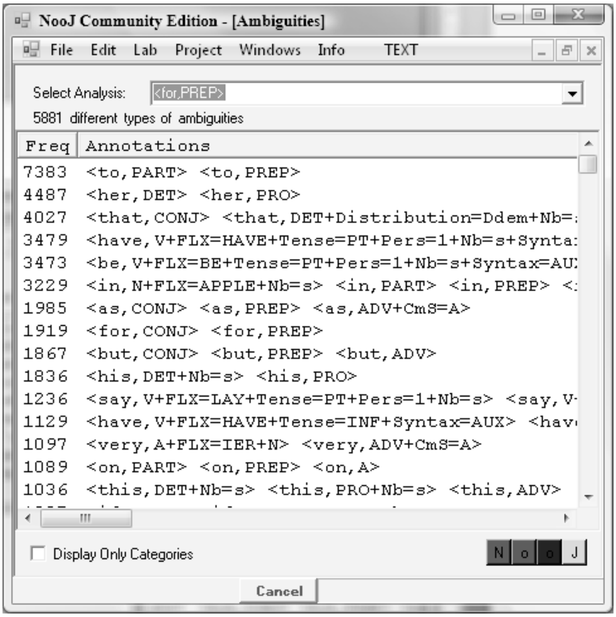
	28	34	38
Nb=p	there,INTJ	are,N+Nb=s+Distribution=Unit	few,DET+Distribution=Dadj+
	there,PRO	be,V+Tense=PR+Pers=2+Nb=s+Syntax=AUX	few,N+Nb=p+Distribution=H
	there,ADV	be,V+Tense=PR+Pers=1+Nb=p+Syntax=AUX	few,PRO+Nb=p

Fig. 6. Editing the Text Annotation Structure

Examining the text's ambiguities

A new tool has been added to NooJ: the possibility of listing ambiguities according to their frequencies, to select one ambiguity and then to apply one solution to the text automatically. The most frequent ambiguous words will be the ones we intend to study in order to disambiguate the text efficiently.

Double-click **Ambiguities** in the text's result window then click the header "**Freq**" to sort all ambiguities according to their frequency.


NooJ Community Edition - [Ambiguities]
File Edit Lab Project Windows Info TEXT
Display Only Categories
Cancel

Select Analysis: <for,PREP>

5881 different types of ambiguities

```
Freq Annotations
7383 <to,PART> <to,PREP>
4487 <her,DET> <her,PRO>
4027 <that,CONJ> <that,DET+Distribution=Ddem+Nb=;
3479 <have,V+FLX=HAVE+Tense=PT+Pers=1+Nb=s+Synta:
3473 <be,V+FLX=BE+Tense=PT+Pers=1+Nb=s+Syntax=AU:
3229 <in,N+FLX=APPLE+Nb=s> <in,PART> <in,PREP> <:
1985 <as,CONJ> <as,PREP> <as,ADV+CmS=A>
1919 <for,CONJ> <for,PREP>
1867 <but,CONJ> <but,PREP> <but,ADV>
1836 <his,DET+Nb=s> <his,PRO>
1236 <say,V+FLX=LAY+Tense=PT+Pers=1+Nb=s> <say,V·
1129 <have,V+FLX=HAVE+Tense=INF+Syntax=AUX> <hav
1097 <very,A+FLX=IER+N> <very,ADV+CmS=A>
1089 <on,PART> <on,PREP> <on,A>
1036 <this,DET+Nb=s> <this,PRO+Nb=s> <this,ADV>
```

Fig. 7. Listing ambiguities according to their frequencies

Here we notice that the word form *for* is ambiguous and occurs 1,919 times in the text. We select the second disambiguation solution (at the top of the window): <for,PREP>. Then we click one of the colored buttons to build the corresponding concordance. Finally, we filter out the rare cases in which *for* is actually a conjunction (I did not find any occurrence in the text *The Portrait of a Lady*) and then click Annotate text (add/remove annotations) to disambiguate the word form.

Conversely, we can double-click Unambiguous Words in the text's result window in order to display the list of all unambiguous words. Usually, the word forms *the, a* and *of* appear most frequently. We can then use frequent unambiguous words as anchors, to remove word ambiguities that occur immediately before or after them. For instance, the word *table* is theoretically ambiguous (it could be a verb), but in the sequences *the table* and *a table* it has to be a noun. Focusing on frequent unambiguous words can help us design new types of disambiguation rules.

Perspectives

In its latest version, NooJ provides a number of tools to deal with all types of ambiguities that are stored in the Text Annotation Structure. Efficient local grammars can remove a large number of ambiguities automatically, and simple enhanced queries can be used to disambiguate a large number of occurrences of a very specific context for a grammatical word. Furthermore, the new ability to directly edit the TAS should help linguists clean up large texts. Moreover, NooJ's users can now display the most frequent ambiguities, as well as the most frequent unambiguous words, which should help them design efficient new local grammars rapidly.

We hope that these new tools will be shortly put to the test: we think that it is very reasonable to build a set of 30+ reliable local grammars that could get rid of 50% of ambiguities, and that semi-automatic and manual tools could be used to get rid of most "annoying" remaining ambiguities in order to build a new set of disambiguated corpora.

References

Silberztein, M. 2003. *NooJ Manual*. Available at http://www.nooj4nlp.net.

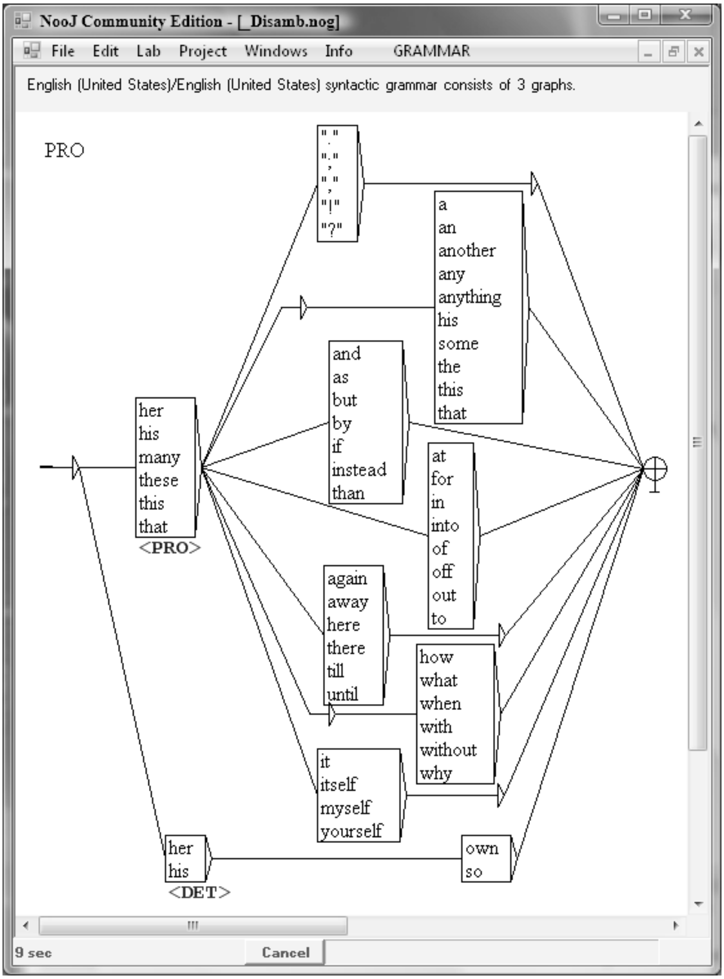
Silberztein, M. 2005. NooJ's Dictionaries. In: *Proceedings of the 2nd Language and Technology Conference*. Poznan.

Silberztein, M. 2007. An Alternative Approach to Tagging. Invited paper In: *Proceedings of NLDB 2007*. LNCS series. Springer. 1-11.

Silberztein, M. 2008. Frozen expressions and discontinuous annotations. In: *Proceedings of NooJ 2007, Barcelona*. Cambridge Schooling Press.

Vietri, S. 2008. The Formalization of Italian Lexicon-Grammar tables in a NooJ Pair Dictionary/Grammar. In: *Proceedings of NooJ 2008, Budapest*. Cambridge Scholars Press.

Appendix A : a disambiguation grammar


NooJ Community Edition - [_Disamb.nog]

English (United States)/English (United States) syntactic grammar consists of 3 graphs.

HFST: Modular Compatibility for Open Source Finite-state Tools

Kimmo Koskenniemi

The Helsinki Finite-State Technology (HFST) aims at facilitating the building of finite-state NLP tools. There are, actually, quite a few packages for basic finite-state transducer (FST) operations available. The goal of the HFST initiative is to partition the task in a way which stimulates both the development of the underlying finite-state calculus and the creation of new compilers and tools for various grammar formalisms.

As seen from the NooJ community, there is a threshold when starting to build a NooJ grammar for yet another language. One might have a well tested and otherwise maintained morphological analyzer available which could be used as a starting point of grammar development. Interfacing existing computer programs for morphological analysis into a host system such as NooJ is not an attractive solution, especially when we expect to interface several different programs. A more coherent FST platform could be a solution for the coexistence of NooJ and various other projects by providing a simple interface between morphological analyzers written in different styles.

CLARIN infrastructure

The EU Commission has funded the preparatory phase of *Common Language Resources and Technology Infrastructure (CLARIN)* which is one of the 34 ESFRI roadmap infrastructures.[1] The aim of CLARIN preparatory phase is to set up the framework of sharing written, spoken and other language materials and tools for processing and using them. It involves at least 100 relevant languages which have a European or national status. The ability to handle a wide array of languages in a uniform framework is quite valuable in the CLARIN context.

[1] EC Grant FP7-RI-2122230 for the three year period 2008-2010. For more information on CLARIN, see *http://www.clarin.eu* and for more information on *European Strategic Forum for Research Infrastructures (ESFRI)*, see *http://cordis.europa.eu/esfri/roadmap.htm*

Finite-state technology

For some time, FST technology has been successfully used in morphological analysis and surface oriented syntactic natural language processing (NLP). Various models and formalisms have been proposed and implemented. We are lacking neither proposals nor implementations. Indeed, lots of software packages for FSTs exist, but the field is fragmented and partitioned which slows down the progress and the reuse of existing results.

There are some commercial and a few dozens (maybe 50-100) open source implementations of the underlying FST calculus. Some of the best quality packages are commercial, such as the XFST of Xerox Corporation[2] and AT&T FSM Library™—Finite-State Machine Library, but being proprietary, the source code is not available if one would like to extend or enhance them. Furthermore, the results produced with these commercial tools cannot be freely used for all purposes. For example, if one creates a transducer lexicon for spelling checking for a minority language, one cannot use it with practical word processors unless one negotiates a commercial license. With large companies this is known to be time consuming and laborious.

There are several (maybe up to 10) matured and maintained open source FST packages and, in addition, dozens of packages which are freely available but whose development and maintenance has ceased. The reuse of any one of the existing FST packages is somewhat difficult and risky in several ways. The first problem is that it is difficult to choose among the available packages because very few comparisons have been made between them and the documentation of the packages is mostly brief, if not insufficient or lacking altogether.

One has to choose, but, once a choice is made, it soon becomes irreversible. Not only the names of functions and data types are idiosyncratic, but the underlying concepts and assumptions differ as well. The application will soon become tied to the particular FST implementation it began to use. There may be nothing wrong with the package which was chosen or with its documentation, but the choice is permanent anyway. Any implementation is based on certain concepts and assumptions, and it is more than likely that different implementations adopt at least slightly different concepts. Even if there might be identical or similar concepts, terms used for them may be different and misleading.

[2] For more information on the Xerox finite-state NLP tool, see *http://www.xrce.xerox.com/competencies/content-analysis/fssoft/home.en.html*.

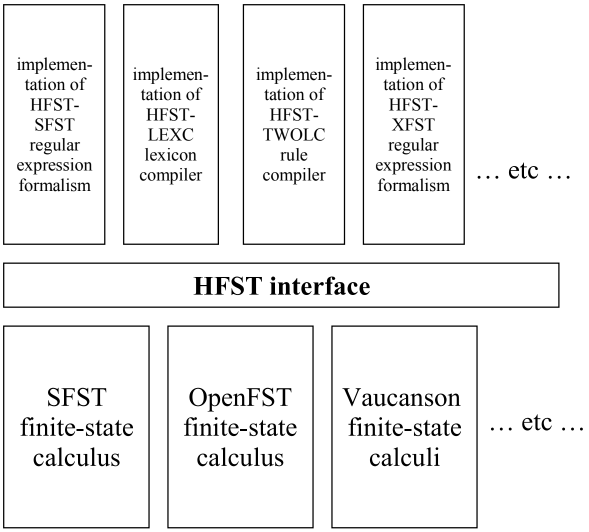
The past efforts which created the existing basic FST technology and NLP tools on top of them have been isolated from each other. Some research groups have built tools for the algebra alone, and others have built NLP tools consisting of a calculus and a compiler for some application oriented rule formalism. Very few research groups have been able to build NLP grammar formalisms on top of the FST calculus created by some other group.

Helsinki Finite-State Toolkit (HFST)

The HFST toolkit effort started in late 2007 at the Department of General Linguistics at the University of Helsinki. The work is directed by PhD Krister Lindén, and the present author and PhD Anssi Yli-Jyrä act as scientific advisors. The programming was done by MA Tommi Pirinen, MA/MSc students Miikka Silfverberg and Erik Axelson. The work was financed by the department as a research project to support the newly started Common Language Resource and Technology Infrastructure (CLARIN) which will need language independent NLP tools for many languages.

The HFST effort has not implemented yet another FST package. Instead, it utilized two existing free, open source finite-state packages: SFST by Helmut Schmid (2005) and OpenFST by M. Riley, J. Schalkwyk, W. Skut, C. Allauzen and M. Mohri (Allauzen et al. 2007). The Stuttgart SFST implements finite-state automata and transducers without weights, and OpenFST with weights. Both are well-known and matured packages.

In order to make these two packages commensurable, HFST has defined and implemented a programming interface for FST concepts and operations. This interface translates the abstract operations into function calls of SFST and OpenFST. The programmer needs not know whether he or she is actually using one or the other. The overall structure of the HFST schema is as follows:


implementation of HFST-SFST regular expression formalism
implementation of HFST-LEXC lexicon compiler
implementation of HFST-TWOLC rule compiler
implementation of HFST-XFST regular expression formalism
… etc …
HFST interface
SFST finite-state calculus
OpenFST finite-state calculus
Vaucanson finite-state calculi
… etc …

Instead of having a monolithic combination of an underlying FST calculus and a tool using it, the HFST interface separates these two components. The HFST interface and the reimplementation of SFST on top of it was the first stage. Then a code to include OpenFST was added to the HFST interface proving the overall validity of the interface. This proved to be somewhat difficult because of the different underlying concepts in the two calculus packages. SFST relies more on a predetermined alphabet and character pair correspondences whereas in OpenFST the role of alphabets and the correspondence pairs is less concrete. The terms used for the two types of FSTs differed, etc. Thus, a HFST terminology had to be defined and established. The naming of HFST interface functions, parameters, variables and data types was also designed to satisfy certain consistent principles. The C++ interface was documented inline using the open source DOXYGEN package.[3]

Evaluating and validating the HFST

The validity of the HFST application programming interface (API) needed to be (informally) verified. This was done by programming some initial

[3] See *http://www.ling.helsinki.fi/kieliteknologia/tutkimus/hfst/* for more information on the HFST, its documentation and downloading its source code.

tools on top of it. The two first tools were reimplementations of the Xerox LEXC and TWOLC compilers which were programmed using only the HFST API. They were then tested using both the SFST and OpenFST package as the underlying calculus. The first versions of these were operational in October 2008 and supported practically all features of the Xerox tools (Beesley and Karttunen, 2003). In order to facilitate the combining of the lexicon transducer and the two-level rule transducers, a special program intersecting composition was also programmed to avoid excessively large intermediate results.

The implementation of the two-level rule compiler (HFST-TWOLC) used the formulas of *Generalized Restriction (GR)* invented by Anssi Yli-Jyrä, which provide a simpler and more general compilation of the two-level than the earlier one reported by Ron Kaplan and Martin Kay (1994).

The tools, HFST-LEXC and HFST-TWOLC, were tested and validated by using two independently created comprehensive morphological lexicons and rule sets: one for Northern Sámi and the other for Lule Sámi, both created by researchers of Sámi languages and a Norwegian project whose aim is to create language technology tools for Sámi languages spoken in Norway.[4]

The testing included the comparison of the outputs of the HFST and the Xerox versions, and large sets of test cases for the two-level rules. After the testing and debugging, the results were identical. The two Sámi languages have a rather complex morphological structure requiring many more rules than e.g. Finnish, whose morphology is often considered to be complicated.

HFST and NooJ

HFST tools and NooJ (Silberztein, 2005) are both intended for processing a wide array of different languages. In addition to offering complementary tools for NLP, HFST and NooJ might benefit from each other in several ways.

Lexicons and other components in FST form which are produced with HFST tools could be reused in NooJ. Thus, there could be an easier threshold for starting projects for syntactic parsing with NooJ if a morphological analyzer already exists in FST form. For some languages, the HFST-LEXC and HFST-TWOLC or similar tools might be a better alternative to describe morphology. For example, in Finnish there are

[4] See *http://giellatekno.uit.no/english.html* for more information on the Sámi language technology project.

2,000 distinct inflectional forms for each noun, and some 12,000-18,000 forms for each verb in the lexicon.

Named entities and new lexemes collected using heuristic guessing mechanisms based on the HFST could be reused in the NooJ framework.

NooJ has been applied to many languages and extensive dictionaries have been created. NooJ has proven to be a feasible tool for corpus processing and annotation. Dictionaries and annotated corpora could be converted and reused e.g. with HFST tools.

NooJ is based on states and transition networks in a framework which is close to the FST paradigm. In principle, NooJ operates on a cascade of transformations according to these networks applied one after the other. While HFST is based on pure FST technology where the algebra of compositions, unions and concatenations is readily available, NooJ augments the transition networks occasionally with registers which moves the networks outside the class of the pure FST calculus. In many cases, still, sets of successive applications of these networks could be interpreted as pure FSTs and their compositions, possibly followed by some additional computation to accomplish the setting and testing of the registers.

References

Allauzen, C., Riley M., Schalkwyk J., Skut W., Mohri M. 2007. Openfst: A general and efficient weighted finite-state transducer library. In: Holub, J., Zdárek, J. (eds.) *CIAA*. LNCS 4783. Berlin, Heidelberg: Springer. 11-23.

Beesley, K., Karttunen, L. 2003. *Finite state morphology*. Stanford (CA): CSLI Publications.

Kaplan, R. M., Kay, M. 1994. Regular models of phonological rule systems. *Computational Linguistics*. Vol. 20. No. 3. 331-378.

Schmid, H. 2005. A programming language for finite state transducers. In: Yli-Jyrä, A., Karttunen L., Karhumäki J. (eds) *FSMNLP*. LNCS 4002. Berlin, Heidelberg: Springer. 308-309.

Silberztein, M. 2005. NooJ: a linguistic annotation system for corpus processing. In: *Proceedings of HLT/EMNLP on Interactive Demonstrations*. Morristown (NJ): Association for Computational Linguistics. 10-11.

Yli-Jyrä, A., Koskenniemi, K. 2004. Compiling contextual restrictions on strings into finite-state automata. In: *The Eindhoven FASTAR Days, Proceedings*. Computer Science Reports. 04/40, Eindhoven, The Netherlands.

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 a 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. A 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 the 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.

The 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 and 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 the Intex or NooJ environment is used repeatedly and iteratively throughout the development process—in the 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 the 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.

The following sections describe the Croatian NERC module and NercFst engine in more detail. We also give insight on Intex and NooJ interacting with other more famous and versatile open source FST engines. The 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 and Tadić 2007). The system is fully implemented in Intex and it is composed of the module for sentence segmentation, a 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