Linguistic richness and technical aspects of an incremental finite-state parser

Hrafn Loftsson*, Eiríkur Rögnvaldsson†

*School of Computer Science, Reykjavik University
Kringlan 1, Reykjavik IS-103, Iceland
hrafn@ru.is

†Department of Icelandic, University of Iceland
Árnagarður við Suðurgötu, Reykjavik IS-101, Iceland
eirikur@hi.is

Abstract

We describe the linguistic richness and the technical aspects of an incremental finite-state parser for Icelandic. We argue that our parser outputs a linguistically rich annotation which in many simple sentences amounts to full parsing. Additionally, we provide arguments for various technical design and implementation decisions regarding the parser. Our description may be used as guidelines for other researchers developing similar parsers.

1. Introduction

Syntactic analysis for natural languages is often divided into two categories: *full parsing*, in which a complete analysis for each sentence is computed, and *partial (or shallow) parsing*, where sentence parts or chunks are analysed without building a complete parse tree. The aim of partial parsing is "to recover syntactic information efficiently and reliably from unrestricted text, by sacrificing completeness and depth of analysis" (Abney, 1996).

In many natural language processing (NLP) applications, it can be sufficient to analyse sentence parts or phrases. This can be the case, for example, in applications like information extraction, machine translation, and some types of grammar checking, in which identification of phrases is more important than a global parse. Additionally, in cases of low quality input or spoken language, a partial parsing method can be more robust than a full parsing method, because of noise, missing words, and mistakes in the input (Li and Roth, 2001).

We have developed an incremental finite-state parser, *IceParser*, for parsing Icelandic text. IceParser, the first parser published for the Icelandic language, is designed to be used both as a stand-alone application and as an integrated part of an NLP toolkit.

In (Loftsson and Rögnvaldsson, 2007a), we briefly described the annotation scheme used by the parser, its individual modules, evaluation results, and error analysis. In this paper, we describe in detail the linguistic richness (in Section 2.) and the technical aspects (in Section 3.) of IceParser.

2. Linguistic richness

In this section, we describe the main features of our annotation scheme and show how it is applied. We argue that through the interplay of phrase structure annotation and syntactic functions annotation, accompanied by relative position indicators for arguments, we obtain a linguistically rich annotation which in many simple sentences amounts to full parsing.

2.1. The annotation scheme

When designing a parser for a natural language, it is important to outline an annotation scheme. In the context of shallow parsing this includes deciding what kind of chunks/phrases and grammatical functions to annotate, and writing guidelines (general principles) on how to perform the annotation (Voutilainen, 1997). Additionally, since "the correct analysis" is not always clear, detailed instructions may be needed where the general principles are not unambiguously applicable.

Voutilainen points out that the annotation scheme (or the grammatical representation, as he calls it) can be specified with the help of a *grammar definition corpus* (GDC). A GDC is a representative collection of sentences, consistently analysed using the guidelines and the detailed instructions. The purpose of the GDC is to "provide an unambiguous answer to the question how to analyse any utterance in the object language" (Voutilainen, 1997). Furthermore, the GDC can be used to help with the development of the parser itself, because the parser should at least be able to annotate correctly the sentences in the GDC.

We have designed a shallow annotation scheme (a thorough description of which can be found in (Loftsson and Rögnvaldsson, 2006)) that follows the dominant paradigm in treebank annotation, i.e. it is "the kind of theory-neutral annotation of constituent structure with added functional tags" (Nivre, 2002). During the design, we focused on making the annotation rich enough to be of use in various NLP applications, in particular, grammar correction. In addition to the annotation scheme, we constructed a GDC, a corpus consisting of 214 sentences, selected from the Icelandic Frequency Dictionary (IFD) corpus (Pind et al., 1991). The selected sentences represent the major syntactic constructions in Icelandic. This GDC was used as the development corpus for IceParser.

It is assumed that input text, to be annotated according to our scheme, has been morphologically tagged using the detailed part-of-speech (POS) tagset of about 700 tags described in (Pind et al., 1991). Although these tags

are morphological in nature, they also carry a substantial amount of syntactic information and the tagging is detailed enough for the syntactic function of words to be more or less deduced from their morphology and the adjacent words (Rögnvaldsson, 2006).
Thus, for instance, a noun in the nominative case can reasonably safely be assumed to be a subject, unless it is preceded by the copula *vera* 'to be' which is in turn preceded by another noun in the nominative, in which case the second noun is a predicative complement. A noun in the accusative or dative case can in most cases be assumed to be a (direct or indirect) object, unless it is immediately preceded by a preposition. As is well known, Icelandic also has accusative and dative subjects, and even some nominative objects (Thráinsson, 2007), but these can easily be identified from their accompanying verbs. We have compiled a list of those verbs which the parser consults.

2.1.1. Phrase structure annotation

According to our annotation scheme, two labels are attached to each constituent. The first one denotes the beginning of the constituent, the second one denotes the end (e.g. [NP ... NP]). The main labels are AdvP, AP, NP, PP and VP – the standard labels used for phrase annotation (denoting adverb, adjective, noun, preposition, and verb phrase, respectively).
Additionally, we use the labels CP, SCP, InjP, and MWE for marking coordinating conjunctions, subordinating conjunctions, interjections, and multiword expressions (of which we have a list), respectively. Furthermore, we use the labels APs and NPs, for marking a sequence of adjective phrases (agreeing in gender, number and case) and noun phrases (agreeing in case), respectively.
Our scheme subclassifies VPs. A finite verb phrase is labelled as [VP ... VP] and consists of a finite verb, optionally followed by a sequence of AdvPs and supine verbs. Other types of VPs are labelled as [VPx ... VPx], where x can have the following values: *i*, denoting an infinitive VP; *b*, denoting a VP which demands a predicative complement (i.e primarily a verb phrase consisting of the verb *vera*); *s*, denoting a supine VP; *p*, denoting a past participle VP; *g*: denoting a present participle VP. Our VPs do not comprise verbal arguments and hence are strictly speaking not verb phrases, but rather bare verbs or verbal clusters. We also distinguish between four kinds of MWEs, i.e. expressions that function as i) a conjunction (MWE_CP), ii) an adverb (MWE_AdvP), iii) an adjective (MWE_AP), and iv) a preposition (MWE_PP).

2.1.2. Syntactic functions annotation

From a linguistic point of view, our constituent structure bracketing is of course rather primitive and in many ways incomplete, compared to the description in (Thráinsson, 2007), for instance. The structure it marks is relatively flat, and, since it is designed for partial parsing, recursiveness is not shown. Thus, many strings that in a complete parse tree would be grouped together in a single multi-level structure are shown as two or more distinct chunks.
Two such examples are shown below (the morphological tags from the input text are underlined):

(1) [NP niðurstöður (*results*) nvfn NP] [NP þessara (*these*, gen.pl.) favfe rannsókna (*research*, gen.pl.) *nvfe* NP]
'The results of this research'

(2) [NP húsin (*the houses*) nhfng NP] [PP í (*in*) aþ [NP fæðingarbæ (*hometown*) nkeþ mínum (*mine*) fekeþ NP] PP] [VPb voru (*were*) sfg3fþ VPb]
'The houses in my hometown were ...'

In (1), the NP *þessara rannsókna* (gen.pl.) 'this research' is a genitive qualifier of the NP *niðurstöður* 'results'. In (2), the PP *í fæðingarbæ mínum* 'in my hometown' modifies the NP *húsin* 'the houses'. The constituent structure bracketing does not indicate any connection between the two NPs in (1), or between the NP and the PP in (2).
The syntactic functions annotation (functional tags), however, substitutes for the lack of hierarchical constituent structure to a considerable extent. We use curly brackets for denoting the beginning and the end of a syntactic function, as carried out by (Megyesi and Rydin, 1999). Four of the function labels we use, *SUBJ, *OBJ, *IOBJ, *COMP, are relatively standard, denoting a subject, an object, an indirect object, and a predicative complement, respectively. To deal with certain important characteristics of Icelandic syntax, we have added four nonstandard labels to mark NPs bearing different functions; *OBJAP, *OBJNOM, *QUAL, *TIMEX, denoting an object of an AP, a nominative object, a genitive qualifier, and a temporal expression, respectively.
Additionally, for some of the syntactic function labels, we use relative position indicators ("<" and ">"). For example, *SUBJ> means that the verb is positioned to the right of the subject, *SUBJ< denotes that the verb is positioned to the left, while *SUBJ is used when the accompanying verb cannot be located, either because it is missing (in gapping structures, for instance) or because the distance between the subject and the verb is more than a parser can cope with. The motivation behind using the indicators is to simplify grammar checking at later stages. Similar indicators are, for example, used in the Constraint Grammar Framework (Karlsson et al., 1995).
By using the syntactic function labels and the relative position indicators, we manage to show the most important relations between phrases. Thus, in a sequence of two adjacent NPs with one of them in the genitive case, as in (1), the genitive NP is marked as *QUAL in order to show that it forms a syntactic unit with the other (governing) NP, as demonstrated in (3):

(3) [NP niðurstöður (*results*) nvfn NP] {*QUAL [NP þessara (*these*, gen.pl.) favfe rannsókna (*research*, gen.pl.) nvfe NP] *QUAL}

In a complete sentence, this string as a whole will be marked as a subject, an object, etc., according to its role in the sentence, as shown in (4):

(4) {*SUBJ> [NP niðurstöður (*results*) nvfn NP] {*QUAL [NP þessara (*these*, gen.pl.) favfe rannsókna (*research*, gen.pl.) nvfe NP] *QUAL} *SUBJ>} [VPb eru (*are*)

sfg3fn VPb] {*COMP< [AP [AdvP mjög (*very*) aa AdvP] óvæntar (*surprising*) lvfnsf AP] *COMP<}
'The results of this research are very surprising'

2.1.3. The output of IceParser

IceParser generates output according to the annotation scheme described above[1]. In many simple sentences, such as in (4), the annotation made using this scheme in fact amounts to a full parse. The phrase structure annotation and the syntactic functions annotation, together with the relative position indicators, give us all the information we need about the structure and the internal dependencies in this sentence. In more complex sentences, of course, the parsing may not be as complete as in this one. This is especially evident in cases of long distance dependencies and in sentences with embedded clauses and clauses with "gaps" of some kind, such as relative clauses.

IceParser makes no attempt at resolving PP attachment ambiguities – all PPs are shown as independent constituents. In the case of a PP following a sentence-initial NP, as in (2) above, we could make use of the fact that Icelandic is a V2 language, which does not allow more than one constituent preceding the finite verb (Thráinsson, 2007). Hence, a PP following an NP in front of a finite verb must be a part of this NP. We could of course show this by closing the function tag of the NP after the PP, but we have not implemented this yet.

2.2. Use of IceParser in grammar correction

Even though IceParser is designed and implemented as a partial parser, we believe that its output is sufficiently detailed to be of great use in many NLP applications, such as in information extraction, grammar correction, and machine translation. Here we will only briefly illustrate its potential use in grammar checking tools.

Among the most error-prone features of Icelandic grammar is morphological agreement and morphological government of various types. Verbs agree in person and number with their subject; predicative adjectives agree in gender, number, and case with the subject of the clause; all inflected words within a noun phrase agree in gender, number, and case; verbs govern the case of their subjects and objects; and so on.

In order to detect errors having to do with agreement or morphological government, it is especially important to nail down the relationship between verbs and their arguments. It has been shown that IceParser does a good job in correctly identifying subjects and objects (F-measure for subjects and objects is 90.5% and 88.2%, respectively (Loftsson and Rögnvaldsson, 2007a)).

In designing the parser, we deliberately chose to make only minimal use of the morphological information furnished by the POS tags in the input text. This was done in order to be able to use the parser in detecting grammatical errors. It is clear that if the parser relies too heavily on the morphological features, errors in the input will both result in parsing errors and also severely undermine the usefulness of the parser in grammar checking. Therefore, the parser mainly uses case features, but other nominal features such as gender and number only in exceptional cases.

Once it has been decided which arguments and predicative complements belong to a certain verb, it can be checked whether the subject NP and the verb agree in person and number. If the verb takes an adjectival complement, it can be checked whether the subject and the complement agree in gender and number.

Since the case features of the morphological tags are used in the parse, it might seem that the parser cannot be used in detecting errors in the case government of verbs, for instance. However, the case feature is mainly used to distinguish between subjects (bearing nominative case, except with certain verbs of which we have a list as mentioned above) and (direct and indirect) objects (bearing oblique case, i.e. accusative, dative, or genitive). Thus, the distinction between the three oblique cases is not crucial for the parse. Most case errors that we find in texts involve some mixup between the oblique cases, rather than between the nominative and one of the oblique cases. Hence, the parser can be used in detecting such errors.

Of course, full parsing, with pronoun resolution etc., would enable us to detect more grammatical errors than our shallow parsing. However, we feel confident that our method will bring us a long way towards useful NLP tools for Icelandic, due to the linguistic richness of the morphological tags and the syntactic annotation scheme.

3. Technical aspects

In this section, we discuss some technical aspects of IceParser. Our aim is to provide arguments for various technical design and implementation decisions, i.e. with regard to the development methodology used, the utilisation of the lexical analyser generator tool JFlex, optimisation, and the integration of IceParser into our NLP toolkit.

3.1. Development methodology

At the very beginning of the parsing project, we decided to base IceParser on the incremental finite-state approach, in which a parser comprises a sequence of finite-state transducers (Grefenstette, 1996; Abney, 1997). The purpose of the transducers is to add syntactic information into running text in an incremental manner. This method is sometimes referred to as the *constructive* approach to distinguish it from the *reductionist* approach by (Koskenniemi et al., 1992) .

The reason for selecting this approach was mainly threefold. First, no treebank exists for Icelandic, and using a data-driven parsing method was therefore not an option. Secondly, earlier incremental finite-state parsing work has proven successful for various languages, e.g. Spanish (Molina et al., 1999), Swedish (Megyesi and Rydin, 1999), German (Müller, 2004), and French (Aït-Mokhtar and Chanod, 1997). Lastly, parsers built using finite-state methods are usually robust and fast, because they are, in fact, just a pipeline of lexical analysers.

Incremental finite-state parsers are developed by specifying patterns, in the form of regular expressions, for matching specific syntactic constructions (substrings) in the input

[1] Strictly speaking, our annotation scheme is independent from IceParser, i.e. the scheme is designed with a general partial parser in mind.

text. Syntactic markers or labels can then be inserted into the input text by associating an action with the appropriate pattern. The syntactic patterns are usually handwritten and thus often demand both computer science knowledge (with regard to regular expressions and finite automata) and linguistic knowledge (of the language being parsed). In fact, our team consists of a computer scientist and a linguist.

Input to IceParser is POS tagged text, using the tagset mentioned in Section 2.1. The parser consists of two main components: a phrase structure module (13 transducers) and a syntactic functions module (9 transducers). The purpose of the modular architecture "is to facilitate the work during development, to allow different uses of the parser and to reflect the different linguistic knowledge that is built into the parser" (Megyesi and Rydin, 1999). In both modules, the output of one transducer serves as the input to the following transducer in the sequence.

Table 1 lists the 22 transducers (in the order in which they are executed) along with a short description of their purpose. The transducers in the upper half of the table belong to the phrase structure module, and the ones in the bottom half belong to the syntactic functions module. Please refer to (Loftsson and Rögnvaldsson, 2007a; Loftsson, 2007b) for a detailed description of all the transducers used in IceParser.

3.2. Utilisation of JFlex

The Xerox Finite-State Tool (XFST) (Karttunen et al., 1996) is often used to develop finite-state parsers (cf. (Megyesi and Rydin, 1999; Aït-Mokhtar and Chanod, 1997)). The XFST includes extensions to the standard regular expression calculus, which simplify the creation of finite-state transducers for syntactic processing. When using the XFST for parser development, the development team defines syntactic patterns, in the form of extended regular expressions, which are then compiled into finite-state transducers. When running the resulting parser (i.e. the set of transducers) on input text, the transducers are interpreted by a run-time engine built into the tool.

We decided not to use the XFST for the development of IceParser. There are mainly two reasons for this decision. First, since the transducers are interpreted by the XFST, its run-time engine needs to be distributed to all parties interested in using the parser. Hence, licensing issues may complicate the distribution of the parser. Secondly, we wanted our parser to be an integrated part of our NLP toolkit (Loftsson and Rögnvaldsson, 2007b), all modules of which are written in Java. In order to simplify the integration of IceParser into the toolkit, and to simplify distribution of the parser, we thus decided to write the parser in a utility which produces Java code.

Our parser is written using the lexical analyser generator tool JFlex (http://jflex.de/). Each transducer is written in a separate specification file, which is compiled into Java code using JFlex. The resulting Java code is a deterministic finite-state automaton, along with actions (Java code) to execute for each matched pattern. The purpose of the actions is to insert syntactic labels into substrings of the input text. The patterns for each transducer are written using the regular expressions language of JFlex. The only non-standard operator of JFlex is $\sim a$, which matches everything up to (and including) the first occurrence of the input matched by a.

As an illustration of the rule and action format of JFlex, consider the following example, taken from the Phrase_MWEP1 transducer which recognises specific MWEs consisting of the preposition *fyrir* followed by specific adverbs:

```
%{
  String Open=" [MWE_PP ";
  String Close=" MWE_PP] ";
%}

AdverbPart = {WS}+{AdverbTag}
PrepPart = {WS}+{PrepTag}

Pair = [fF]yrir{PrepPart}(aftan|austan
       |framan|neðan|norðan|ofan|sunnan
       |utan|vestan){AdverbPart}
%%
{Pair} {out.write(Open+yytext()+Close);}
```

The code included in %{ and %} is copied directly into the generated Java source code.

Two regular definitions[2], *AdverbPart* and *PrepPart*, define the adverb part and the preposition part of the <preposition, adverb> pair, respectively. For example, the adverb part consists of one or more white spaces ({WS}+) followed by an *AdverbTag*. *AdverbTag* is a name defined in a special file, which is included by most of the transducers (*PrepPart* is defined similarly):

```
AdverbTag = aa[me]?{WS}+
```

i.e. the letters *aa* optionally followed by the letters *m* or *e* and postfixed with one or more white spaces. Finally, the name *Pair* is defined as the preposition *fyrir* followed by specific adverbs.

Actions are included inside curly brackets. Thus, when the generated lexical analyser recognises the pattern *Pair* the action is simply to put the appropriate brackets and labels around it (obtained by the function call yytext()). For example, for the MWE *fyrir aftan* 'behind' the result is:

(5) [MWE_PP fyrir ao aftan aa MWE_PP]
(*ao* and *aa* are the POS tags denoting preposition and adverb, respectively.)

Note that the action described above is a simple implementation of the *left to right longest match markup* replace operator, described in (Karttunen et al., 1996). This operator is a part of the XFST, in which it is specified by using the following syntax:

```
A @-> B ... C
```

A transducer using this operator then inserts the strings (markers) B and C around the longest string matched by regular expression A.

[2]Regular definitions are a sequence of definitions of the form: d_i -> r_i, where each d_i is a distinct name and each r_i is a regular expression which may refer to names $d_1 \ldots d_{i-1}$.

Name	Purpose
Phrase_MWE	Marks MWEs: common bi- and trigrams.
Phrase_MWEP1	Marks MWEs: specific <preposition, adverb> pairs.
Phrase_MWEP2	Marks MWEs: specific <adverb, preposition> pairs.
Phrase_AdvP	Marks adverb, conjunction, and interjection phrases.
Phrase_AP	Marks adjective phrases.
Case_AP	Adds case information to adjective phrases.
Phrase_APs	Groups together a sequence of adjective phrases.
Phrase_NP	Marks noun phrases.
Phrase_VP	Marks verb phrases.
Case_NP	Adds case information to noun phrases.
Phrase_NPs	Groups together a sequence of noun phrases.
Phrase_PP	Marks prepositional phrases.
Clean1	Corrects special kind of annotation errors.
Func_TIMEX	Marks temporal expressions.
Func_QUAL	Marks genitive qualifiers.
Func_SUBJ	Marks subjects.
Func_COMP	Marks complements.
Func_OBJ	Marks direct objects.
Func_OBJ2	Marks indirect objects.
Func_OBJ3	Marks dative objects of complement adjective phrases.
Func_SUBJ2	Marks "stand-alone" nominative noun phrases.
Clean2	Cleans up, e.g. superfluous white spaces.

Table 1: A brief description of all the transducers.

3.3. Optimisation

In the first version of IceParser (and presumably in most incremental finite-state parsers), the output file of one transducer is used as the input file to the next transducer in the sequence. This version processes about 14,900 word-tag pairs per second (running on a Dell Precision M4300, Intel Core 2 Duo CPU, 2.2 GHz).

Note that, by writing the parser in Java/JFlex, we have full control of the source code. This has enabled us to build an optimised version of IceParser. In the optimised version, instead of making the transducers read and write to files, we make them read from, and write directly to, memory.

The following Java function *parse* illustrates how the output of one transducer is used as input to the next transducer in the sequence, without reading and writing to files.

```
0) parse(String text) {
...
1) StringReader sr=new StringReader();
2) StringWriter sw=new StringWriter();
3) advp=new Phrase_AdvP(sr);
4) ap=new Phrase_AP(sr);
5) advp.yyreset(sr);
6) advp.parse(sw);
7) sr=new StringReader(sw.toString());
8) sw=new StringWriter();
9) ap.yyreset(sr);
10)ap.parse(sw);
...
}
```

Line 0) is the signature of the function *parse*, which is called once for every line in a file containing the text to be parsed. Lines 1), 2), 7), and 8) create instances of the *StringReader* and *StringWriter* Java classes. Lines 3) and 4) create instances of the Phrase_AdvP and Phrase_AP classes, whose source files were automatically created by the JFLex tool from a corresponding regular expression specification file (as discussed in Section 3.2.). Lines 5) and 9) reset the corresponding lexical analyser to read from a new input stream. Lines 6) and 10) call the parse method in the corresponding transducers (see below). Moreover, lines 6), 7), and 9) show how the output of the Phrase_AdvP transducer is used as input to the Phrase_AP transducer.

The *parse* method of the transducers tries to match the input to its patterns and carry out the associated action. Its implementation is simple:

```
parse(java.io.Writer _out)
{
  out = _out;
  while (!zzAtEOF)
       yylex();
}
```

Note that methods starting with the letters *yy* (like yyreset() and yylex()) and variables starting with the letters *zz* (like zzAtEOF) are automatically generated by the JFlex tool. The *out* variable, which is an instance of the Writer class, is used in the actions to transduce the output (as shown in the last line of the code example for the Phrase_MWEP1 transducer in Section 3.2.).

This optimised version of IceParser annotates the whole POS tagged IFD corpus, consisting of 590,297 word-tag pairs (36,922 sentences), in just over 23 seconds. This is equivalent to about 25,200 word-tag pairs per second, resulting in a speed increase of about 70% compared to the first version of the parser.

3.4. Integration into our NLP toolkit

Since IceParser is written in Java, we were able to integrate it easily into our NLP toolkit, *IceNLP*. The toolkit works in the following manner. First, the input text is tokenised. Secondly, sentence segmentation is carried out. Thirdly, POS tagging for each input sentence is performed (using *IceTagger*, a linguistic rule-based tagger (Loftsson, 2007a)), and, lastly, each POS tagged sentence is partially parsed with IceParser.

The optimised version of IceParser is used in IceNLP. A POS tagged sentence is not written to an output file, but is instead fed directly to IceParser in the manner described in Section 3.3. The result is an efficient combined POS tagging and partial parsing utility[3].

4. Conclusion

In this paper, we have described the linguistic richness and the technical aspects of IceParser, an incremental finite-state parser for Icelandic. We discussed the linguistic richness of the output generated by the parser and argued that for many simple sentences the output amounts to full parsing. Additionally, we described technical aspects of the parser, in particular, various design and implementation details. Our description may be used as guidelines for other researchers developing similar parsers.

5. Acknowledgements

Thanks to the Institute of Lexicography at the University of Iceland, for providing access to the IFD corpus used in this research.

The development of IceParser was partly supported by the Icelandic Research Fund, grant 060010021, "Shallow parsing of Icelandic text".

6. References

S. Abney. 1996. Part-of-Speech Tagging and Partial Parsing. In K. Church, S. Young, and G. Bloothooft, editors, *Corpus-Based Methods in Language and Speech*. Kluwer Academic Publishers.

S. Abney. 1997. Partial Parsing via Finite-State Cascades. *Natural Language Engineering*, 2(4):337–344.

S. Aït-Mokhtar and J.-P. Chanod. 1997. Incremental Finite-State Parsing. In *Proceedings of Applied Natural Language Processing*, Washington DC, USA.

G. Grefenstette. 1996. Light Parsing as Finite State Filtering. In *Proceedings of the ECAI '96 workshop on "Extended finite state models of language"*, Budapest, Hungary.

F. Karlsson, A. Voutilainen, J. Heikkilä, and A. Anttila. 1995. *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*. Mouton de Gruyter, Berlin, Germany.

L. Karttunen, J.-P. Chanod, Grefenstette, G., and A. Schiller. 1996. Regular expressions for language engineering. *Natural Language Engineering*, 2(4):305–328.

K. Koskenniemi, P. Tapanainen, and A. Voutilainen. 1992. Compiling and using finite-state syntactic rules. In *Proceedings of the* 14^{th} *International Conference on Computational Linguistics*, Nantes, France.

X. Li and D. Roth. 2001. Exploring Evidence for Shallow Parsing. In *Proceedings of the* 5^{th} *Conference on Computational Natural Language Learning*, Toulouse, France.

H. Loftsson and E. Rögnvaldsson. 2006. A shallow syntactic annotation scheme for Icelandic text. Technical Report RUTR-SSE06004, Department of Computer Science, Reykjavik University.

H. Loftsson and E. Rögnvaldsson. 2007a. IceParser: An Incremental Finite-State Parser for Icelandic. In *Proceedings of NoDaLiDa 2007*, Tartu, Estonia.

H. Loftsson and E. Rögnvaldsson. 2007b. IceNLP: A Natural Language Processing Toolkit for Icelandic. In *Proceedings of Interspeech 2007, Special Session: "Speech and language technology for less-resourced languages"*, Antwerp, Belgium.

H. Loftsson. 2007a. Tagging Icelandic Text using a Linguistic and a Statistical Tagger. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics*, Rochester, NY, USA.

H. Loftsson. 2007b. *Tagging and Parsing Icelandic Text*. Ph.D. thesis, University of Sheffield, Sheffield, UK.

B. Megyesi and S. Rydin. 1999. Towards a Finite-State Parser for Swedish. In *Proceedings of NoDaLiDa 1999*, Throndheim, Norway.

A. Molina, F. Pla, L. Moreno, and N. Prieto. 1999. APOLN: A Partial Parser of Unrestricted Text. In *Proceedings of SNRFAI99*, Bilbao, Spain.

F-H. Müller. 2004. Annotating Grammatical Functions in German Using Finite-State Cascades. In 20^{th} *International Conference on Computational Linguistics*, Geneva, Switzerland.

J. Nivre. 2002. What kinds of trees grow in Swedish soil? A Comparison of Four Annotation Schemes for Swedish. In *Proceedings of the* 1^{st} *Workshop on Treebanks and Linguistic Theories*, Sozopol, Bulgaria.

J. Pind, F. Magnússon, and S. Briem. 1991. *Íslensk orðtíðnibók [The Icelandic Frequency Dictionary]*. The Institute of Lexicography, University of Iceland, Reykjavik, Iceland.

E. Rögnvaldsson. 2006. The Corpus of Spoken Icelandic and its Morphosyntactic Annotation. In *Treebanking for Discourse and Speech. Proceedings of the NODALIDA 2005 Special Session on Treebanks for Spoken Language and Discourse*, Copenhagen, Denmark.

H. Thráinsson. 2007. *The Syntax of Icelandic*. Cambridge University Press, Cambridge.

A. Voutilainen. 1997. Designing a (Finite-State) Parsing Grammar. In E. Roche and Y. Schabes, editors, *Finite-State Language Processing*. MIT Press.

[3]Please visit `http://nlp.ru.is` for tagging and parsing Icelandic text.