Tagging and Parsing Icelandic Text

Hrafn Loftsson

Submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy

Department of Computer Science
University of Sheffield, UK
May 2007

Contents

List of Figures

List of Tables

Acknowledgements

First, I would like to thank my supervisor, Professor Yorick Wilks. He was very supportive during the time I stayed in Sheffield and made numerous valuable comments and suggestions throughout this work. Secondly, I thank my other thesis committee members, my chair, Dr. Anthony Simons, for being so enthusiastic about my work, and my advisor, Dr. Mark Hepple, for his important feedback on my work on tagging.

The Institute of Lexicography at the University of Iceland kindly provided access to the tagged corpora used in this thesis. Moreover, I thank the Icelandic newspaper Morgunblaðið for giving access to text used in this research.

I thank the Icelandic Research Fund for partly supporting the parsing project "Shallow parsing of Icelandic text", which is the foundation for the parsing part of this thesis. Furthermore, I thank Professor Eiríkur Rögnvaldsson, at the University of Iceland, for collaboration and project management in this parsing project.

I thank my employer, Reykjavik University, for giving me the flexibility needed for completing this work.

Last, but not least, I thank my family for their tolerance during the last three years.

Hrafn Loftsson
Reykjavik, May 2007

Publications

A part of this thesis has appeared in the following publications:

- Loftsson, H. and Rögnvaldsson E. 2007. 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.
- Loftsson, H. and Rögnvaldsson, E. 2007. IceParser: An Incremental Finite-State Parser for Icelandic. In *Proceedings of NoDaLiDa 2007*. Tartu, Estonia.
- Loftsson, H. 2007. Tagging Icelandic Text using a Linguistic and a Statistical Tagger. In *Proceedings of Human Language Technologies 2007: The Conference of the North American Chapter of the ACL*. Rochester, NY, USA.
- Loftsson, H. and Rögnvaldsson, E. 2006. A shallow syntactic annotation scheme for Icelandic text. Technical Report RUTR-SSE06004, Department of Computer Science, Reykjavik University.
- Loftsson, H. 2006. Tagging Icelandic text: an experiment with integrations and combinations of taggers. *Language Resources and Evaluation*, **40(2)**:175–181.
- Loftsson, H. 2006. Tagging a Morphologically Complex Language Using Heuristics. In T. Salakoski, F. Ginter, S. Pyysalo, and T. Pahikkala (eds.), *Advances in Natural Language Processing, 5^{th} International Conference on NLP, FinTAL 2006, Proceedings*. Turku, Finland.
- Loftsson, H. 2006. Tagging Icelandic text: A linguistic rule-based approach. Technical Report CS-06-04, Department of Computer Science, University of Sheffield.

Abstract

Natural language processing (NLP) is a very young discipline in Iceland. Therefore, there is a lack of publicly available basic tools for processing the morphologically complex Icelandic language.

In this thesis, we investigate the effectiveness and viability of using (mainly) rule-based methods for analysing the syntax of Icelandic text. For this purpose, and because our work has a practical focus, we develop a NLP toolkit, *IceNLP*. The toolkit consists of a tokeniser, the morphological analyser *IceMorphy*, the part-of-speech tagger *IceTagger*, and the shallow parser *IceParser*.

The task of the tokeniser is to split a sequence of characters into linguistic units and identify where one sentence ends and another one begins.

IceMorphy is used for guessing part-of-speech tags for unknown words and filling in tag profile gaps in a dictionary.

IceTagger is a linguistic rule-based tagger which achieves considerably higher tagging accuracy than previously reported results using taggers based on data-driven techniques. Furthermore, by using several tagger integration and combination methods, we increase substantially the tagging accuracy of Icelandic text, with regard to previous work.

Our shallow parser, *IceParser*, is an incremental finite-state parser, the first parser published for the Icelandic language. It produces shallow syntactic annotation, using an annotation scheme specifically developed in this work. Furthermore, we create a grammar definition corpus, a representative collection of sentences annotated using the annotation scheme.

The development of our toolkit is a step towards the goal of building a Basic Language Resource Kit (BLARK) for the Icelandic language. Our toolkit has been made available for use in the research community, and should therefore encourage further research and development of NLP tools.

Part I

Introduction

Chapter 1

Introduction

It has been predicted that, in the future, the main method of communication between humans and computers (or other processing devices) will be natural language (NL), in both spoken and written forms. This, indeed, seems evident; since we humans communicate most easily with one another using NLs, why should we not use this innate capability of ours to communicate with computers as well? Moreover, communicating with devices using NL will enable the average person to interact with computers without demanding any special skills or training.

In fact, today, humans already use NL for interacting with various systems, i.e. writing queries to search engines and entering credit-card numbers using voice. Furthermore, various systems have already been developed to process NLs for some particular task, e.g. grammar correction, information extraction, corpus annotation and machine translation.

Language technology (LT) is the underlying technology that enables us to use NLs for communicating with computers and enables computers to process, understand and interpret NLs. LT is a multidisciplinary field which requires knowledge from different disciplines like linguistics, psychology, engineering and computer science. Generally, it can be said, however, that LT comprises two main subcategories: speech technology and natural language processing (NLP).

Speech technology, which uses data in spoken form, includes two main subfields, i.e. speech synthesis and speech recognition. The former is concerned with producing human speech, whereas the latter addresses recognition of human speech.

In contrast, NLP, which is the subject of this thesis, deals with automatic

processing of NLs in written or textual form. NLP is among the oldest disciplines of computer science — work on machine translation started as early as in the 1950s. The field of NLP has grown significantly the last ten years, mainly due to the increased amount of textual data available for processing on the World Wide Web (www). Processing, understanding and interpreting NLs is, however, not a trivial task. The reason is that NLs are ambiguous, i.e. a particular word or a word segment can have different interpretations depending on the context.

Ambiguity is, usually, categorised into two types: lexical and syntactic. Lexical ambiguity arises when a particular word can have more than one meaning. For example, in the above sentence part "*a particular word or a word segment*", the word "*segment*" is lexically ambiguous, because it can either be a noun or a verb. Syntactic ambiguity, on the other hand, is caused by the grammatical structure of a sentence. For example, in the widely used illustrative sentence "*I saw the man with a telescope*", a syntactic ambiguity arises because it is not clear from this sentence alone whether "*I*" used "*a telescope*" to see "*the man*" or whether "*the man*" was seen holding "*a telescope*" (this kind of syntactic ambiguity can be represented as the problem of prepositional phrase attachment).

In this thesis, we will address lexical ambiguities and analyse the grammatical structure of sentences using Icelandic as our working language. We will develop a NLP toolkit, *IceNLP*, for processing Icelandic text. *IceNLP* consists of a tokeniser/sentence segmentiser, a morphological analyser, *IceMorphy*, a part-of-speech (POS) tagger, *IceTagger*, for resolving lexical ambiguities, and a shallow parser *IceParser* to recover syntactic information (without resolving syntactic ambiguities). Notice that we are only concerned with syntax issues with regard to the Icelandic language, but leave out semantics and pragmatics for future work.

1.1 Motivation

The Icelandic population is very small; about 300,000 people live in Iceland and probably less than 400,000 people speak the language worldwide. Consequently, one can reflect upon whether it is worth the effort to make the Icelandic language suitable for use in an information technology (IT) society. In this context, making the language suitable means, for example, translating software packages from English (or other languages) to Icelandic and build-

ing the necessary tools that are a prerequisite for efficient LT research and development.

Why not simply use the English language for IT in Iceland? Icelanders start to learn English at the age of ten, British and North-American TV programs are common, and the proficiency of the Icelandic people in English is generally considered good. Nevertheless, the question above has to be answered negatively.

First, one can consider patriotic views. In addition to preserving its Sagas, the Icelandic nation is well known for the preservation of its language. It is a common opinion in the small Icelandic population that the language has to be preserved. Without preservation of the mother tongue in all fields of national life, Icelandic literature and other cultural particularities will gradually disappear. Instead of incorporating new English names for inventions and processes into the language, as many nations do, the Icelandic community has the tendency, unforced, to make new Icelandic words for things as they come up. As people gradually communicate more with computers using NLs, the lack of Icelandic translations and the scarcity of NLP and speech recognition tools will become a hindrance. Without it, Icelanders will need to use English to communicate with computers. In the long run this will endanger the language the Icelandic people care so much about.

Secondly, realistic political positions should be considered. Modern IT systems have already become European Union (EU) multilingual and localised. One of the main reasons for this is the strong position of big language communities in Europe, like France and Germany. In these countries there is a strong policy to use the mother tongue in all areas of IT. Moreover, the EU has encouraged multilingual use, e.g. through its Multilingual Information Society Programme (MLIS 1996-1999; `http://www.cordis.lu/ist/98vienna/xmlis.htm`). Thus, we can assume that future IT systems will have multilingual support as one of their design criteria and, hence, adaptation of the systems to the particularities of the Icelandic language should neither be very difficult nor extremely costly.

1.1.1 Language Technology in Iceland

LT in Iceland was practically non-existent only a few years ago. In 1999, a report was written by a commission for the Icelandic Ministry of Education, Science and Culture (MESC) on the status of LT in Iceland. The following points, borrowed from the report (Arnalds 2003), were amongst

the conclusions:

- Far fewer language technology tools were available for Icelandic than for more widely used languages in neighbouring countries.
- Individual information technology companies did not have the resources to perform basic research in language technology.
- Basic research in language technology in Iceland hardly existed.

To react to this situation, the commission proposed that the government supported an effort to improve the state of LT in Iceland. The propositions were as follows (Ólafsson 1999):

- Corpora should be built and made accessible for research and development of LT tools.
- A special fund should be established to support research in the field of LT.
- Companies should be sponsored in order to develop LT tools.
- Educational programs in the field of LT should be established.

As can be inferred from the points above, LT in Iceland in 1999 was in its infancy. LT tools were scarce, no funds existed for supporting the field and no university offered educational programs in LT. Icelandic corpora were scarce in 1999 but some, in the interest of LT, had been constructed at the Institute of Lexicography (IL) at the University of Iceland. Most notable was the *Icelandic Frequency Dictionary* (*IFD*) published in 1991 (Pind et al. 1991).

After the establishment of the *The Icelandic Language Technology Fund* in 2001 (by MESC), a small number of projects have been carried out in the field of LT by universities and private companies (Menntamálaráðuneytið 2004). Moreover, a MA program in LT has now (2002) been established by the University of Iceland.

1.2 Basic Language Resource Kit

The first item in the list of propositions above is related to the concept of BLARK — *Basic Language Resource Kit*. The concept of the BLARK was first introduced in the ELRA (European Language Research Association) newsletter in 1998 (Krauwer 1998). In the newsletter, it was stated that a BLARK for a language should contain a specification of:

- the minimum general text corpus required to be able to do any precompetitive research for the language at all, say (as an arbitrary example) 10 million words of recent newspaper text, annotated according to generally accepted standards
- something similar for a spoken text corpus
- a collection of basic tools to manipulate and analyse the corpora
- a collection of skills that constitute the minimal starting point for the development of a competitive NL/Speech

In later writings, the BLARK has been defined as "the minimal set of language resources that is necessary to do any precompetitive research and education at all" (Krauwer et al. 2004). Note that the contents of a BLARK can vary from language to language, and that over time it may gradually evolve as new technologies emerge. However, the underlying idea has been to make a general BLARK definition applicable in principle to all languages, since such a definition saves time and effort when a BLARK is developed for new languages.

An important point, with regard to BLARK, is that "in order to make a BLARK for a language maximally impactful, the language resources of which it consists should be easily and reliably accessible, inexpensive and usable" (Krauwer et al. 2004).

1.3 Key research hypothesis

During the last ten to fifteen years or so, data-driven methods (DDMs) have been used extensively to develop syntax analysis components for various languages. The purpose of DDMs is extracting information from data automatically and thus (mostly) eliminating the need for human intuition in the

analysis of the data. In contrast, linguistic rule-based methods (LRBMs) are based on hand-crafted rules, which are built using human linguistic knowledge. The proliferation of DDMs can be attributed to *i)* the increased availability of annotated corpora which these methods can "learn" from; *ii)* the fact that most DDMs are language independent and can thus be applied to various languages; and *iii)* the effectiveness (i.e. high accuracy) of many of these methods[1].

However, in this thesis, we hypothesise that LRBMs provide an effective (with regard to performance) and a viable (with regard to resource/manpower needs) approach for developing syntax analysis components for Icelandic. More specifically, we hypothesise that:

- **Due to data sparseness, a higher tagging accuracy can be obtained by a linguistic rule-based tagger when tagging Icelandic text, than achieved by a state-of-the-art data-driven tagger. Moreover, this can be achieved without using an enormous effort in development of the tagging system**.

- **The use of a finite-state parsing method for a morphologically complex language, with a relatively free word order, like Icelandic, is effective, and additionally, that an enormous effort is not needed in the development of a finite-state parser for the language in order to obtain good results.**

The motivation for the first hypothesis above is discussed in Section 5.3.1, and for the second hypothesis in Section 6.1. Here, we provide a brief summary:

- Previous part-of-speech tagging experiments for Icelandic using DDMs have demonstrated inferior results compared to related languages. We presume that this is due to data-sparseness, i.e. the size of the tagset used in relation to the size of the training data. Hence, we presuppose that a higher tagging accuracy can be obtained by developing a method which uses linguistic knowledge. Moreover, in order not to spend an enormous time in developing local rules (see Section 5.3.4), we will also use heuristics (see Section 5.3.5) in the disambiguation process.

[1] In Chapter 2, we discuss some data-driven methods in the context of part-of-speech tagging and parsing.

- No syntactically annotated corpus of Icelandic text exists and developing a data-driven parser for Icelandic is thus currently not an option. Shallow parsing, as opposed to full parsing, is sufficient in many NLP systems. Finite-state parsing, which relies on a set of local syntactic patterns, is one form of shallow parsing which has proven to be effective for various languages. Despite the fact that Icelandic has a relatively free word order, its rich case system should help when writing patterns for grouping words into phrases and identifying syntactic functions. Hence, we presuppose that using a finite-state parsing method for Icelandic will be viable and result in an effective parser.

1.4 Thesis contribution

Since NLP is a very young discipline in Iceland, basic tools like a tokeniser, a sentence segmentiser, a tagger and a parser are not yet available for the research community. The lack of basic tools does, of course, hinder development of more sophisticated tools like software for grammar correction, information extraction, question-answering, corpus annotation, and machine translation.

The main contribution of this thesis is the following:

- An NLP toolkit, *IceNLP*, has been developed for the purpose of processing Icelandic text. The development of this toolkit is a step towards the goal of building a BLARK for the Icelandic language.

  Our toolkit consists of the following sequentially applied modules[2]:

  1. **Preprocessor**
     (a) Sentence segmentation
     (b) Tokenisation
  2. **POS tagger — *IceTagger***
     (a) Morphological analyser — *IceMorphy*
        i. Dictionary lookup
        ii. Unknown word guessing
        iii. Tag profile gap filling

[2] Each of these modules can also be used as a stand-alone unit. The toolkit as a whole can be tested by visiting `http://nlp.ru.is`.

   (b) Disambiguation
      i. Local rules
      ii. Heuristics
3. **Finite-state parser — *IceParser***
   (a) Phrase structure module
   (b) Syntactic functions module

Each of the above modules will be described in detail in Chapters 5 and 6.

- A morphological analyser/unknown word guesser for Icelandic, *IceMorphy*, has been developed and evaluated. *IceMorphy* includes a *tag profile gap filler* module. In the literature, we have not found equivalent modules for other languages. No other such morphological analyser for Icelandic is publicly available.

- A linguistic rule-based tagger for tagging Icelandic text, *IceTagger*, has been developed and evaluated. It achieves considerably higher accuracy than previously reported results using taggers based on data-driven techniques. This is the first linguistic rule-based tagger developed for tagging Icelandic text. The development time of our linguistic rule-based tagging system was only 7 man months which can be considered a short development time for a linguistic rule-based system. We are not aware of another comparable system, developed in such a short time frame, which outperform state-of-the-art data-driven methods which use supervised learning.

- By using several tagger integration and combination methods the tagging accuracy of Icelandic text has been increased substantially, compared to previously reported results.

- A shallow syntactic annotation scheme has been developed for Icelandic text. Moreover, a grammar definition corpus, a representative collection of sentences annotated using the annotation scheme, has been compiled[3]. This is the first syntactically (semi-automatically) anno-

[3]The development of the annotation scheme and the grammar definition corpus was jointly carried out with Professor Eiríkur Rögnvaldsson (University of Iceland), in the project "Shallow parsing of Icelandic text", funded by the Icelandic Research Fund in 2006.

tated corpus for Icelandic text.

- A shallow parser, *IceParser*, which produces shallow syntactic annotation for Icelandic text, has been developed and evaluated. This is the first publicly available parser for the Icelandic language. Evaluation shows that *IceParser* obtains results that are comparable to related languages. On the other hand, a comparison with related languages indicates that our results are good. The overall time spent on development of the parsing system (including the annotation scheme and the grammar definition corpus) was about 1 man year.

- Our work and evaluation results supports our hypothesis, i.e. that LRBMs provide an effective and a viable approach for developing syntax analysis components for Icelandic.

1.5 Outline

The remainder of this thesis is divided into eight chapters.

Chapter 2 presents introductory material on NLP, with emphasis on language analysis, i.e. tokenisation, morphological analysis, POS tagging and (shallow) syntactic parsing.

Chapter 3 describes the Icelandic language with the purpose of giving the reader "a feel" for the morphological complexity of the language and basic syntactic issues.

Chapter 4 gives an overview of the corpora and the tagset used in this research.

Chapter 5 describes our tagging system, the main component of *IceNLP*. Individual components of the system are described and discussed with regard to previous work.

Chapter 6 describes the other part of *IceNLP*, the parsing system.

Chapter 7 presents evaluation results on tagging Icelandic text. Results are given for *IceTagger* and compared to results obtained by data-driven taggers. Moreover, this chapter demonstrates how various combination methods can be used to improve the tagging accuracy.

Chapter 8 presents evaluation results on parsing Icelandic text, both for constituent structure and syntactic functions.

Finally, we conclude in Chapter 9 with an overview and a discussion on future work.

Chapter 2

Natural Language Processing

This chapter presents some introductory material on NLP. First, the task and the goal of NLP are defined. Then, individual stages of NLP applications are described along with a review of the literature.

2.1 Introduction

The discipline of NLP concerns itself with the design and implementation of computer systems that communicate with humans using natural language. The goal of NLP is to build systems that can analyse, understand and generate natural languages. The main emphasis is on practical development of language processing systems and on building reusable modules that can be glued together to construct NLP applications[1].

"Traditionally, work in natural language processing has tended to view the process of language analysis as being decomposable into a number of stages, mirroring the theoretical linguistic distinction drawn between syntax, semantics and pragmatics"[2] (Dale 2000). The syntax stage can, further, be viewed as a sequence of the finer grained stages: tokenisation, lexical analysis and syntactic analysis.

[1] Sometimes, a distinction is made between NLP and Language Engineering (LE). Cunningham (2000) defines NLP as "the part of science of computation whose subject-matter is computer systems that process human language", and LE as "the discipline of engineering software systems that perform tasks involving processing human language".

[2] This is, for example, reflected in the chapter organisation of the well known textbook on Speech and Language Processing by Jurafsky and Martin (2000).

The first stage of a typical NLP application is preprocessing or tokenisation, in which the stream of input text is broken into tokens and sentences.

The second stage, lexical analysis, can, broadly speaking, be defined as determining the lexical features of the individual words in the text. Which lexical features are identified depends on the application at hand, but, typically, NLP applications need information on morpho-syntactic features, i.e. morphological and POS information.

Syntactic analysis, or parsing, follows the lexical phase. Syntactic analysis is the task of analysing sentence structure and the dependencies between its individual parts. Parsing is, usually, not a goal in itself, but is a necessary step for the next stage, semantic analysis.

In order to "understand" NL texts it is necessary to assign meaning to individual sentences. This is the task of semantic analysis, the fourth stage. Semantics of NL have been less studied than issues of syntax, probably because semantic analysis depends on the aforementioned stages, which need to be "mastered" first, and, additionally, because assigning meaning to sentences might, in fact, be a more difficult step.

Finally, we have the stage of pragmatic analysis, whereby the meaning of the utterance or text in context is determined.

In the following sections, we will restrict the description of the NLP stages to the work presented in this thesis — namely preprocessing, lexical analysis (i.e. POS tagging and morphological analysis) and parsing.

2.2 Preprocessing

Most work in NLP requires preprocessing steps which performs sentence segmentation and word tokenisation. Sentence segmentation identifies where one sentence ends and another one begins. The task of tokenisation is to split a sequence of characters into simple tokens (linguistic units) like words, numbers and punctuation marks.

The above preprocessing steps are often considered trivial but, in fact, can present surprising problems. In the case of sentence segmentation, it is, for example, not sufficient to search for the standard end-of-sentence characters, periods, exclamation points and question marks, because sometimes semicolons, colons, dashes and commas can serve as an end-of-sentence marker. Furthermore, each of these punctuation characters can serve several purposes. For example, a period can serve as an end-of-sentence marker, an

abbreviation, a decimal point, etc.

Like sentence segmentation, tokenisation might at first seem like a simple task, but there are a number of things that need to be accounted for. Special care needs to be taken when tokenising punctuation characters, abbreviations and multipart words, and multiword expressions may need to be accounted for. Moreover, since text commonly contains mark-ups, like HTML, SGML or XML codes, text filtering needs to be carried out. A good coverage of sentence segmentation and word tokenisation can be found in (Grefenstette and Tapanainen 1994, Palmer 2000).

In the context of programming language compilation, tokenisation and lexical analysis are used interchangeably. When programming languages are compiled, the first phase consists of lexical analysis whose goal is to identify the minimal units of analysis (i.e. tokens) to constitute the starting point for the second phase, syntactic analysis. In contrast, in the context of NLP, lexical analysis has a broader meaning than tokenisation alone, since it often includes morphological analysis and disambiguation (POS tagging; see Section 2.3). On the one hand, the goal of lexical analysis of a natural language is the same as for a programming language, i.e. to prepare text for syntactic analysis. On the other hand, there are many important differences between lexical analyses of the two kinds of languages (Silberztein 1997).

First, there is the size of the vocabulary used. Programming languages typically use only dozens of keywords, but the vocabulary for natural languages can consist of tens or hundreds of thousands of words, compiled into a dictionary. Secondly, because words in texts are commonly inflected and because of the existence of unknown words (i.e. words unknown to the dictionary at hand), morphological analysis is required when processing natural languages. Finally, words in natural languages can be ambiguous at various levels, but programming languages are designed to be unambiguous. For example, the ambiguity at the lexical level, i.e. the fact that a given word can have various POS, calls for a special disambiguation phase, POS tagging, before syntactic analysis is carried out.

2.3 Part-of-speech tagging

POS tagging is the task of labelling words with the appropriate word class and morphological features (therefore, POS tagging is often referred to as morpho-syntactic tagging). POS tagging can be seen as a mapping from

sentences to strings of labels. The string used as a label is called a *tag*, the set of labelling strings is called a *tagset*, and a program which performs tagging is called a *tagger*. Tagging text is needed for several NLP tasks, e.g. grammar correction, syntactic parsing, information extraction, question-answering and corpus annotation.

To illustrate, consider the following Icelandic sentence: "*gamli maðurinn borðar kalda súpu með mjög góðri lyst*" (old man-the eats cold soup with very good appetite). Next, we show this sentence with possible tags accompanying each word ("_" is used as a separator between tags):

*gamli/***lkenvf** *maðurinn/***nkeng** *borðar/***sfg3en_sfg2en**
*kalda/***lhenvf_lkfosf_lveosf_lkeþvf_lheþvf_lheovf_lheevf**
*súpu/***nveo_nveþ_nvee** *með/***aþ_aa**
*mjög/***aa** *góðri/***lveþsf** *lyst/***nveþ_nveo_nven**

In Section 4.2, we will describe the semantics of these tags in detail, but, very briefly, the tags denote word class information and morphological features, like gender, number and case.

Since a word can be ambiguous in its POS (i.e. a word can have multiple possible tags), the main function of a tagger is to remove ambiguity. Many taggers perform this task by, first, introducing ambiguity (lexical phase) and, then, performing disambiguation (disambiguation phase). The former, a relatively easy task, consists of introducing the *tag profile* (the set of possible tags) for each word, both known and unknown words. This can be carried out with the help of a pre-compiled dictionary and an unknown word guesser, whose function is to guess the tag profile for words not known to the dictionary. The disambiguation task is more difficult, since, in order to disambiguate, a tagger needs to consider the context in which a particular word appears.

After disambiguation, our illustrative sentence has the following tags:

*gamli/***lkenvf** *maðurinn/***nkeng** *borðar/***sfg3en** *kalda/***lveosf** *súpu/***nveo**
*með/***aþ** *mjög/***aa** *góðri/***lveþsf** *lyst/***nveþ**

There are two main methodologies for disambiguation: the data-driven and the linguistic rule-based (constraint-based) approach. In the data-driven approach a pre-tagged training corpus is used to automatically obtain information to be used later during disambiguation. The disambiguation information acquired can, for example, be in the form of statistics or rules. In

contrast, a linguistic approach uses hand-crafted rules or constraints to eliminate inappropriate POS tags depending on context.

The earliest taggers used hand-crafted rules for assigning tags to words based on character affixes of words and on the basis of the tags of the surrounding words (Klein and Simmons 1963, Cherry 1980). The tagset used in the Cherry tagger was small (only 10 labels) since the purpose of the tagger was only to label each word with its word class. Tagging words with only the word class is in most cases of limited use for NLP applications. Therefore, it is common, nowadays, to use a tagset consisting of tens or hundreds of different tags — the larger the tagset the more precise the POS analysis for each word.

There has been a tendency to develop data-driven taggers (DDTs) in the last ten to fifteen years. The DDTs are language and tagset independent and usually simpler to develop than hand-crafted rule-based taggers because no (or limited) human effort is needed for rule writing. Additionally, developing a linguistic rule-based framework, able to compete with DDTs, has been considered a difficult and time-consuming task (Brill 1992, Samuelsson 1994, Voutilainen 1995). A different opinion has, however, been expressed in (Chanod and Tapanainen 1995).

A number of different data-driven tagging methods have been developed. Well known methods include, for example, probabilistic methods based on a Markov model (Church 1988, Cutting et al. 1992, Brants 2000), a maximum entropy approach (Ratnaparkhi 1996), a transformation-based learning approach (Brill 1992; 1995a) and a memory-based approach (Daelemans et al. 1996). About 96-97% accuracy (the ratio of corrected tags to the total number of tags) has been achieved with these taggers for English text (using the Penn Treebank tagset; see next section). It is widely accepted that, in order to achieve 100% tagging accuracy, a tagger would need to incorporate semantic or world knowledge (see Abney (1996) for a rationale). A comprehensive systematic evaluation of DDTs for other languages than English, can, for example, be found in a study using the Swedish language (Megyesi 2002). In this study, a tagging accuracy of 89.3%-93.6% was obtained, using a tagset of 139 tags.

In contrast to DDTs, linguistic rule-based taggers (LRBTs) are developed with the purpose of tagging a specific language using a particular tagset. One of the better known linguistic rule-based methods is the Constraint Grammar (CG) framework (Karlsson 1990, Karlsson et al. 1995) which has been applied to several languages. The English Constraint Grammar parser,

EngCG, was "the first serious linguistic competitor to data-driven statistical taggers" (Samuelsson and Voutilainen 1997).

It has been shown that combining taggers will often result in a higher tagging accuracy than achieved by individual taggers. The reason is that different taggers tend to produce different errors and the differences can be exploited to yield better results. A number of different combination methods exists, e.g. simple voting, weighted voting and stacking (van Halteren et al. 2001), as well as combinations using linguistically motivated rules (Borin 2000).

2.3.1 Tagsets

"Historically, the most influential tagsets have been the ones used for tagging the American Brown corpus (the Brown tagset) and the series of tagsets developed at the University of Lancaster, and used for tagging the Lancaster-Oslo-Bergen (LOB) corpus and more recently, the British National corpus (CLAWS1 through CLAWS5)" (Manning and Schütze 2002, chap. 4).

Recently, however, the most common tagset used for tagging English text is the relatively small Penn Treebank tagset consisting of 45 tags (Marcus et al. 1993). For comparison, the University of Lancaster CLAWS7 tagset, for example, consists of 137 tags, but the size of the set is due to lexical differentiation rather than inflectional complexity.

Naturally, tagsets are dependent on the underlying language, and, obviously, a larger tagset will make more fine-grained distinctions. Because of inflectional properties, the tagsets used for the Scandinavian languages are considerably larger than those most often used for the English language. For example, 139 tags were used in the previously mentioned Swedish tagging experiment (Megyesi 2002) and the Danish Parole tagset consists of 151 tags (Hardt 2001). The Icelandic tagset used in our research contains about 660 tags (see Section 4.2).

It is common that tagsets for highly inflectional and agglutinative languages consists of 400-1000 tags, e.g. the tagsets for Hungarian, Czech, Romanian, and Slovene (Hajič 2000).

It has generally been assumed that larger tagsets result in lower tagging accuracy, since, in that case, the tagger simply has more tags to choose from for each word. However, the relationship between tagset size and tagging accuracy is vague. Research has shown that, in fact, the use of a larger tagset can, at least for inflectional languages, result in higher accuracy (Elworthy

1995). The reason is that a more detailed tagset may help the tagger select tags for words that are in some way connected. For example, this might be true in case of a subject-verb agreement and nominal feature agreement in phrases.

2.3.2 Dictionaries

Dictionaries used by taggers commonly consist of key-value pairs[3]. Each key is a word form and the corresponding value is the tag profile for the given word form. A typical dictionary format is thus (using "_" as a separator between tags):

$$
\begin{aligned}
&w_1 = t_{11}\_t_{12}\_\ldots\_t_{1s_1} \\
&w_2 = t_{21}\_t_{22}\_\ldots\_t_{2s_2} \\
&\vdots \\
&w_n = t_{n1}\_t_{n2}\_\ldots\_t_{ns_n}
\end{aligned}
$$

Here, n is the number of word forms in the dictionary, w_i is word form number i, t_{ik} is the k^{th} tag for w_i, and s_i is the number of tags for word form i ($i = 1 \ldots n$). In some cases, the tags for a given word form are sorted according to frequency — e.g. the most frequent tag might appear first in the tag profile for a given word form. To illustrate, the following is a record from an Icelandic dictionary for the word "*við*" (see Appendix B for explanation of the individual tags):

við=ao_fp1fn_ap_aa_nkeo

Most taggers use dictionaries containing tens of thousands of word forms. For larger dictionaries, and when speed is of prime importance, a more compact dictionary format may be suitable. A known method is to represent the dictionary as a deterministic-finite automaton (DFA), which ensures both low access time and small storage space.

A dictionary in the "standard" format described above can be encoded in the following manner. First, a special kind of tree, called a *trie* is built. The branches in the trie are labelled with letters and the leaves are labelled with

[3]Following Ciura and Deorowicz (2001), we distinguish between a dictionary and a lexicon. The latter does not have any values associated with the keys.

a tag profile. Then, the resulting trie is minimised into a DFA (for example, by using an algorithm described by Ciura and Deorowicz (2001)).

2.3.3 Evaluating taggers

The purpose of an evaluation of a tagger (or an NLP tool in general) is to give an indication of how well or badly the tagger performs on new unseen texts. Moreover, an evaluation can be used to compare the performance of different systems. In that case, it is of prime importance that the systems are evaluated using the same training and test data, and the same tagset as well.

The tagger evaluation method described below is the one that has become a standard, and is generally used in the literature. First, a tagged corpus is split into a training set and a test set. The training set is then used to train data-driven taggers and/or build dictionaries used by either data-driven or linguistic rule-based taggers[4]. Thereafter, the test set (the unseen part) is used to evaluate the performance of the taggers. This method is usually referred to as the *holdout validation method*, presumably because the test data is held out such that the taggers are not allowed to use it for training.

The evaluation figures obtained by a holdout method can potentially be dependent on the exact split into training and test data. In order to filter out this dependance, the so-called *n-fold cross-validation method* is normally used. In this method, the data set (i.e. the corpus) is divided into n subsets, and the holdout method is repeated n times. Each time, one of the n subsets is used as the test set and the other $n-1$ subsets are put together to form a training set. The evaluation figures presented are then the average figures over all n trials.

In the literature, one finds several measures that are used to indicate the performance of a tagger (van Halteren (1999) presents a good overview). The most common are *accuracy*, *error rate*, *ambiguity rate*, *precision* and *recall*.

Accuracy (also called correctness) measures how many tokens assigned

[4]Indeed, dictionaries for many linguistic rule-based taggers/morphological analysers (e.g. taggers based on the CG framework) have been built by hand or derived from other machine readable resources than corpora. In that case, comparison with a data-driven tagger becomes less reliable.

by a tagger receive a contextually correct tag:

$$accuracy = \frac{\#\text{ of correctly tagged tokens}}{\text{total }\#\text{ of tokens}} \tag{2.1}$$

The *error rate* of a tagger is defined as $1.0 - accuracy$.

The *ambiguity rate* measure is often used when a tagger does not perform full disambiguation, i.e. in the case of ambiguous tagging. Ambiguity rate measures the average number of tags assigned to each token. The less the better, i.e. assuming ambiguous tagging it is preferable that the ambiguity rate is low:

$$ambiguity\ rate = \frac{\text{total }\#\text{ of tags}}{\text{total }\#\text{ of tokens}} \tag{2.2}$$

Additionally, the measures *precision* and *recall* (originating in information retrieval) are often used to indicate the performance of ambiguous taggers. For tagging, *precision* measures how many of the tags suggested by a tagger are actually correct, and *recall* measures how many of the correct tags are actually proposed by the tagger:

$$precision = \frac{\#\text{ of correct tags produced by tagger}}{\text{total }\#\text{ of tags produced by tagger}} \tag{2.3}$$

$$recall = \frac{\#\text{ of correct tags produced by tagger}}{\#\text{ of correct tags}} \tag{2.4}$$

In addition to obtaining a single value for precision and a single value for recall for a tagger, these measure can be used to indicate the performance of a tagger for each individual tag. In that case, precision measures how many of the tokens tagged X are tagged X correctly, and recall measures how many of the tokens that ought to have the tag X are indeed tagged X.

Note that when a tagger performs full disambiguation, ambiguity rate=1 and accuracy=precision=recall.

2.3.4 Data-driven methods

Data-driven tagging methods use machine learning to automatically derive a language model from, usually, hand-annotated corpora. The main advantage of the data-driven approaches is that they are language independent and no (or limited) human effort is needed for derivation of the model. On the other hand, the disadvantage is, first, that a pre-tagged corpus, which can be very

time consuming to construct, is essential for training, and, second, that a limited context size is normally used for disambiguation (e.g. three words in the case of a trigram tagger; see below).

In the machine learning literature, distinction is made between *supervised* learning methods and *unsupervised* methods. Supervised methods use annotated training data to learn a function which can be used to predict the value of any input data. In the context of tagging, the annotated training data is a tagged corpus and the function is a *classifier* which predicts the tags of input words.

In contrast, unsupervised methods do not require annotated training data, but, rather, induce a model to fit observations. For tagging, the unsupervised models use computational methods to automatically induce word groupings (i.e. tagsets) and based on those automatic groupings, to either calculate the probabilistic information needed by stochastic taggers (Cutting et al. 1992) or to induce the context rules needed by rule-based systems (Brill 1995b).

In this thesis, we only experiment with supervised tagging methods. In the following sections, we describe four types of supervised data-driven methods: two probabilistic methods, a transformation-based learning method and a memory-based learning method. A number of papers have been published in which taggers based on these methods have been evaluated and compared. Various languages have been used, e.g. English (Brill 1995a, Ratnaparkhi 1996, Daelemans et al. 1996, Brants 2000), Dutch (van Halteren et al. 1998), Hungarian (Kuba et al. 2004), Icelandic (Helgadóttir 2004), Slovene (Džeroski et al. 2000), and Swedish (Megyesi 2002).

2.3.4.1 Probabilistic tagging

One type of a DDT is a probabilistic (stochastic) trigram tagger based on a Markov model. The states of the model represent pairs of tags and the model emits words each time it leaves a state. A trigram tagger finds an assignment of POS to words by optimising the product of lexical probabilities and contextual probabilities. Lexical probability is the probability of observing word i (w_i) given POS j ($p(w_i|t_j)$) and contextual probability is the probability of observing POS i given k previous POS ($p(t_i|t_{i-1}, t_{i-2}, \ldots, t_{i-k})$; $k = 2$ for a trigram model). The probabilities of the model are estimated from a training corpus using maximum likelihood estimation. Thereafter, a sentence can be tagged automatically by assigning it the tag sequence which receives the

highest probability by the model.

The main advantage of a probabilistic tagger is that the necessary statistics can be automatically acquired and no or very little linguistic knowledge is built into the system. The disadvantage of this approach is, however, that the knowledge is hidden in large tables of statistics which can make improvements to the tagger difficult. The *TnT* tagger (Brants 2000) is an example of an effective and an efficient trigram tagger.

Another type of a probabilistic DDT, *MXPOST*, is based on a maximum entropy approach (Ratnaparkhi 1996). It generates probability distributions like other statistical methods and, additionally, uses a binary feature representation to model tagging decisions which can be compared to rules in rule-based methods. A feature f_i asks a yes/no question about a particular history, h_i (a sequence of words and tags), available when predicting tag t_i. The feature f_i, which restricts the value of t_i, encodes any information that can be used to predict t_i, such as the spelling of the current word or the identity of tags and/or words. The goal of the model is to maximise the entropy of a distribution subject to certain feature constraints.

A tagger based on the maximum entropy approach has similar advantages/disadvantages as a tagger based on a Markov model.

2.3.4.2 Transformation-based learning

A third type of a successful DDT is the transformation-based learning (TBL) approach (Brill 1992; 1995a). This approach is rule-based, but the transformation rules (which change a tag X to tag Y) are not hand-crafted, but, rather, automatically acquired from a pre-tagged corpus.

The tagger initially assigns each word its most likely tag without regard to context[5]. At each iteration, the current assignment is compared to the pre-tagged text and a transformation is learnt which results in the greatest reduction of errors. The current assignment is updated using the learnt transformation before the next iteration starts. At the end, the result is an ordered list of transformations that can be applied to the output of the initial assignment to increase the accuracy. New unseen text is then tagged by first applying the initial annotator followed by the application of the ordered transformation rules.

[5]Thus, a transformation-based tagger is an example of a tagger which does not start by introducing the whole tag profile for each word.

This method has the advantage over the probabilistic methods that the learnt linguistic information is represented in a concise and perspicuous manner in the form of learnt rules. The disadvantage with the standard TBL approach is its long training time when used on large-sized corpora. Each iteration step in the algorithm requires that a transformation rule candidate is applied to the whole corpus to calculate the effects of applying the rule and the current assignment of the corpus updated before the next iteration starts.

A fast version of TBL (*fnTBL*), which speeds up the training time of the standard version without sacrificing performance, has been developed by Ngai and Florian (2001). Instead of applying a new rule to the whole corpus, the *fnTBL* method identifies which previously generated rules and associated words (samples) are influenced by the new rule.

The transformation rules are stored in memory at each iteration (instead of being regenerated each time), together with two sets, $G(r)$ and $B(r)$. $G(r)$ is the set of samples to which the rule r applies and changes them to the correct classification. $B(r)$ is the set of samples to which the rule r applies and changes a correct classification to an incorrect one. The objective function for r is $f(r) = |G(r)| - |B(r)|$. Given a new learnt rule b, the *fnTBL* method identifies the rules r, for which at least one of $G(r)$, $B(r)$ is modified by the application of rule b. Thus, only samples that change $f(r)$ need to be checked.

A disadvantage with the standard transformation-based approach is that the tagging of new sentences is inefficient. For each individual rule acquired during the learning phase, the algorithm scans the input from left to right while attempting to match the rule. "The algorithm treats each rule as a template of tags and slides it along the input, one word at a time" (Roche and Schabes 1997). Moreover, the rules can potentially interact, i.e. one rule can undo a change carried out by a previous rule. Roche and Schabes have shown that the sequential application of rules can be represented by a single non-deterministic finite-state transducer[6]. This non-deterministic transducer can then be converted into a deterministic one, which makes the resulting tagger very efficient.

[6] A finite-state transducer is a finite-state automaton that consumes input and produces output on each state transition.

2.3.4.3 Memory-based learning

The memory-based learning tagger (*MBT*) (Daelemans et al. 1996) uses similarity-based reasoning when making tagging decisions. The method is called memory-based because the tag of a word is determined from the most similar cases held in memory.

Each case in memory consists of a word, its left and right context, and the corresponding tag for the word in that context. The memory, which is composed during the learning phase, consists of examples represented as vectors of feature values with an associated tag label. A new sentence is tagged by selecting, for each word in the sentence and its context, the most similar case(s) held in memory and a single tag is selected for the word by applying a nearest neighbour technique.

Memory-based learning is computationally expensive since the feature value vector representing a word (and its context) to be tagged has to be compared to the feature value vectors of each case held in memory. A compression formalism, *IGTree*, has been developed to make the method more effective (Daelemans et al. 1996). The formalism restricts the search to match a new case to those that have the same feature value at given "important" features.

2.3.5 Linguistic rule-based methods

In contrast to DDTs, LRBTs are developed with the purpose of tagging a specific language using a particular tagset. The purpose of the rules is either to assign tags to words depending on context or, in the more common reductionistic approach, to remove illegitimate tags from words based on context.

The advantage of LRBTs is that they do not rely (to the same extent as DDTs) on the existence of a pre-tagged corpus and rules can be written to refer to words and tags in the entire sentence. The construction of a LRBT can, however, be a time-consuming task since the rules are usually hand-crafted and the number of rules is often in the hundreds or thousands. As an illustration of the number of rules used in LRBTs, one can mention the Swedish CG project, in which 2,100 rules are used to remove ambiguity (Birn 1998) and EngCG-2 (English Constraint Grammar version 2), a project developed over several years, with 3,600 constraints (rules) (Samuelsson and Voutilainen 1997). The Norwegian Constraint Grammar project, which took

seven man labour years, is another example of a labour intensive linguistic rule-based project (Hagen et al. 2000). A prerequisite for the effectiveness of a CG parser is the construction of an extensive dictionary and a morphological analyser, both of which demand large resources.

The input tokens to a CG parser are morphologically analysed words. Each word is labelled with tags indicating POS, inflection, derivation, etc., as well as with syntactic function tags. The constraints, which are formulated on the basis of extensive corpus studies, are then used to remove as many inappropriate tags from these words as possible.

A disadvantage of the CG Framework is "that constraints cannot be generalised, but have to be stated in a case by case fashion" (Hinrichs and Trushkina 2002). This is probably the reason for the large number of rules usually developed under this framework.

In the German parsing scheme GRIP, two kinds of disambiguation rules are used: *ordinary disambiguation rules*, which eliminate readings for a single token based on local context, and general *double reduction rules*, which reduce readings of a sequence of tokens, e.g. inside noun phrases (Hinrichs and Trushkina 2002). Our tagger uses a similar idea, i.e. *local rules* for initial disambiguation and *global rules or heuristics*, to force feature agreement for further disambiguation (see Section 5.3.5).

The error rate of the EngCG-2 system has been reported as an order-of-magnitude lower than the error rate of a statistical tagger (Samuelsson and Voutilainen 1997). However, it is important to note that the EngCG-2 system does not perform full disambiguation and the results are, thus, only presented for the same ambiguity levels[7] in the two taggers. Hence, it is not clear what the difference in the error rate between the two systems would be if both would perform full disambiguation. Moreover, it can be inferred from this research that, when testing the EngCG-2 system, the unknown word ratio (which is not specified in the paper) is less than the corresponding ratio (2.01%) when testing the statistical tagger. The reason is that the former system uses a large hand-compiled dictionary, but the latter a dictionary derived from a training corpus.

By making a syntactic parser eliminate ambiguity, which standard CG rules were not able to handle, a LRBT was able to achieve about 99% accuracy, using full disambiguation, when tagging English text (Voutilainen 1995). However, one can argue that using a LRBT along with a syntactic

[7] An ambiguity level is a predefined ambiguity rate (average number of tags per word).

parser to eliminate remaining ambiguity does not constitute a fair comparison with DDTs because a DDT could also been implemented in such a way.

Another tagger comparison, using the French language, demonstrated superior performance of a constraint-based tagger over a statistical one (Chanod and Tapanainen 1995). In this research, a tagged corpus was not available and thus the statistical tagger was trained and tuned using suitable amount of untagged text (i.e. unsupervised learning was used). Since the tagger was not trained using a tagged corpus, so-called biases were set to instruct the tagger what is likely (and what is not) in a given context. The biases, which serve as initial values before training, represent kinds of lexical probabilities and contextual probabilities. The same amount of time was spent on developing rules for the rule-based tagger as was spent on the training phase of the statistical tagger. It is, however, important to bear in mind that statistical taggers based on unsupervised learning are generally less accurate than those based on supervised learning (Merialdo 1994).

2.3.6 Morphological analysis

Morphological analysis is the task of recognition and production of word forms. In the recognition phase, word forms are parsed and recognised as being a part of a particular morphological class. In the production (generation) phase, word forms are produced given a base form belonging to a particular morphological class.

The computational model *two-level morphology*, by Koskenniemi (1983; 1984), is undoubtedly the best known morphological model used in NLP. It is a language independent model, based on the formalism of generative phonology, in which the morphology of a language is described with a set of rewriting rules. A rewriting rule changes or transforms one symbol into another symbol. "The rules start from an underlying lexical representation, and transform it step by step until the surface representation is reached" (Koskenniemi 1984). In contrast to unidirectional rules in generative phonology, the rules in two-level morphology (which are represented by a finite-state transducer) are bidirectional, i.e. they can be used both for recognition and production of word forms.

In addition to rewriting rules, the two-level model needs a lexical component, which lists indivisible words and morphemes in their underlying form. The two components work together to perform both production and recognition.

In the context of POS tagging, where only the phase of recognition is necessary, morphological analysis is often referred to as *unknown word guessing*. During tagging, unknown word forms need to be recognised as belonging to particular morphological classes, denoted by one or more possible tags.

2.3.6.1 Unknown word guessing

The main problem in the lexical phase of tagging is guessing the tag profile for unknown words. In fact, the tagging accuracy of unknown words considerably affects the total tagging accuracy. Continuously extending the dictionary, to minimise the number of unknown words, is not practical because new words are constantly being introduced into a language. Therefore, in order to develop a high accuracy tagger, a good quality unknown word guesser is essential.

The methods used by the aforementioned DDTs (discussed in Section 2.3.4) for guessing unknown words are rather different. The unknown word guesser of the *TnT* trigram tagger uses ending analysis based on a probability distribution. The distribution for a particular ending is generated from words in the training corpus that share the same ending of some predefined length. Interestingly, the module assumes that endings of infrequent words in the training data are a better approximation for unknown words, rather than endings of frequent words. Additionally, the module generates a different distribution around capitalised words from those not capitalised.

The maximum entropy tagger *MXPOST* uses a similar assumption regarding the distribution of unknown words as the *TnT* tagger, i.e. that rare words in the training data are similar to unknown words in the test data, with respect to how affixes help predict their tags. The feature templates used when predicting tags for unknown words include prefixes and suffixes of length ≤ 4, as well as information regarding whether the word contains uppercase letters, hyphen or a number.

The transformation-based tagger *fnTBL* uses a method, originally proposed by (Brill 1995a), which automatically learns cues to predict the most likely tag for unknown words. First, an unknown word is labelled as a proper noun if capitalised and a common noun otherwise. Secondly, transformation templates are used to learn rules which change the initial tag X to another tag Y. These templates make reference to any string of characters up to a bounded length and, typically, refer to suffixes or prefixes of words.

The memory-based tagger *MBT* allows one to encode the last X letters

of unknown words as separate features. Additionally, one can encode the first letter as a feature and thus provide information about prefixes and capitalisation. Thereafter, context information is added to the case at hand in a similar way as is done with known words.

Since LRBTs are developed in order to tag a particular language, they usually include a morphological component/unknown word guesser specifically tailored to the language at hand. Most unknown word guessing modules use morphological/compound analysis and/or ending analysis. The difference between morphological analysis and ending analysis is that the former bases its analysis on morphologically related words already known to the dictionary, whereas the latter bases its analysis solely on a word's ending. Not surprisingly, research has shown that morphological analysis is more accurate than ending analysis (Mikheev 1997, Nakov et al. 2003). This can be explained by the fact that morphologically related words share the same stem (the common part shared by all forms) as the given unknown word, whereas ending analysis does not take the stem into account. Therefore, ending analysis, generally, produces more tag candidates than morphological analysis.

The Morphy tool consists of a morphological analyser and a POS tagger for German (Lezius 2000). The morphological analyser performs inflectional analysis by determining the stem by a reversing morphological process. The analyser requires a dictionary of stems and their corresponding inflection types. If a dictionary entry is found for a stem candidate, the stem's inflection is generated according to the inflection type (morphological class). The original word form is then matched against this list of generated words. The tool also contains an ending analyser (used if the morphological analysis fails) which relies on German suffix statistics.

Nakov et al. (2003) describe ending-guessing rules for classification of unknown German nouns. The system accepts raw text as input and produces a list of unknown nouns along with their stem and morphological class. The rules are automatically generated in a similar manner as in Mikheev's system (see below). From a training text, a set of potential rules is generated depending on endings, the corresponding morphological class and frequencies. Potential rules are then scored to filter out rules that are sensitive to data sparseness.

Mikheev (1997) presents a technique for automatic rule induction for unknown word guessing. The method guesses possible POS tags for unknown words based on their starting and ending segment. Morphological rules (both

prefix and suffix rules), as well as ending guessing rules, are statistically induced using a general-purpose dictionary mapped to a particular tagset, and a raw corpus. When the induced guessing rules were incorporated into existing taggers, the tagging accuracy improved significantly.

2.3.7 Combination methods

Tagger combination methods are a means of correcting for the biases of individual taggers. It has been shown that combining taggers will often result in a higher tagging accuracy than achieved by individual taggers (Brill and Wu 1998, van Halteren et al. 1998; 2001, Sjöbergh 2003). The reason is that different taggers tend to produce different errors, and the differences, "provided they are complementary and systematic" (Borin 2000), can be exploited to yield better results. For a tagger combination pool, it is thus important to select taggers that are based on different language models.

A number of different combination methods exists, e.g. *simple voting*, *weighted voting* and *stacking* (van Halteren et al. 2001), as well as combinations using *linguistically motivated* rules (Borin 2000).

In simple voting, each tagger gets an equal vote when voting for a tag and the tag with the highest number of votes is selected. In weighted voting, more weight is given to taggers that have shown high accuracy, e.g. a tagger known to produce high overall accuracy gets more weight when voting, or a tagger known to produce high precision for a given class of tags gets higher weight for that class (see (van Halteren et al. 2001) for a detailed discussion).

A combination method is called stacking when a learner (classifier) is trained to produce tags, using the output of one or more taggers for training (Wolpert 1992). Apart from the training phase, two other important features of stacking differentiate it from voting. First, stacking does not necessarily select one of the tags proposed by the individual taggers and, second, when deciding on a combined tag, the learner can use contextual information. It has been shown that stacking methods can result in higher accuracy compared to using voting methods (van Halteren et al. 2001).

Combining taggers using linguistically motivated rules is similar to weighted voting in the sense that the relative strength of a given tagger, in a particular linguistic context, is utilised in the combination. Furthermore, by using linguistically motivated rules, tags can be chosen with regard to context, as is the case for stacking.

Adding more taggers to a combination pool normally improves the tagging

accuracy, even if the new taggers are not very good (Sjöbergh 2003). This is, however, generally not the case when adding a tagger which is in some sense similar to another tagger(s) in the pool (because similar taggers make similar errors).

When combining N taggers, the time needed to process the text to be tagged can be expected to be at least a factor N greater compared to using a single tagger. However, using a combination method is worthwhile when the processing time is much less important than the additional gained tagging accuracy, e.g. in the case of corpus annotation.

A combined tagger is, moreover, of important use when estimating and locating errors in an annotated corpus. We will discuss this, with regard to the *IFD* corpus, in Section 7.4.4.2.

2.3.8 Integration methods

Tagger integration is a task which is not discussed frequently in the literature. By tagger integration, we mean making one tagger use a feature or a functionality of another tagger (or a morphological analyser), in such a way that the resulting system runs like a single tagger.

In order to integrate functionality of one tagger into another, it is often necessary to have access to the source code of one or both taggers — this fact, indeed, is probably the reason why integration methods are infrequently discussed.

In Section 7.3, we will show how a rule-based method can be integrated with a statistical method, and *vice verse*, in order to improve the tagging accuracy on Icelandic text. Moreover, we show how our morphological analyser can be integrated with two data-driven taggers.

2.4 Parsing

Syntactic analysis, or parsing, is the task of analysing sentence structure and the dependencies between its individual parts. A parser is a program which performs parsing. Usually, the input to a parser is morphologically disambiguated word-tag pairs (i.e. the input has been tagged) and the output is a structural description of the sentences and tags denoting various dependencies. The syntactic structure is described using a grammar which depends on a particular grammar formalism.

Syntactic analysis for natural languages is often divided into two categories. On the one hand, there is full parsing, in which a complete parse tree is constructed for each sentence. On the other hand, there is shallow (partial) parsing, where sentence parts or chunks are analysed without building a complete parse.

2.4.1 Full parsing

The purpose of full parsing is to compute a complete syntactic analysis for a given sentence using a particular grammar formalism. The best known grammar formalism is the context-free grammar (CFG), which associates the conventional phrase structure tree with each sentence. Well known parsing algorithms that can parse all context-free languages include the CYK (Cocke-Younger-Kasami) algorithm (Younger 1967) and the Earley algorithm (Earley 1970).

The standard version of the CYK algorithm recognises languages defined by grammars in Chomsky normal form[8]. The algorithm uses dynamic programming to build a recognition matrix which lists, for each substring of a string to be parsed, all the non-terminal symbols N which generate the substring. This matrix can be used to recognise if a particular string is in the language or not and, additionally, with small modifications, to build a parse matrix which includes information on how to derive the string.

The Earley algorithm performs a top-down search and can handle left-recursive grammars. It is similar to the CYK algorithm in the sense that for each input symbol scanned a state is constructed which represents the condition of the recognition process at that point in the scan. The algorithm uses a data structure called a chart which is used for looking up parsed constituents (partial parse trees) and, hence, re-parsing already seen constituents is avoided.

A number of other different grammar formalisms have been developed or adopted from linguistics to facilitate full parsing. Among the most used are the constraint-based grammar models, e.g. Head-Driven Phrase-Structure Grammar (HPSG) (Pollard and Sag 1994) and Lexical Functional Grammar (LFG) (Kaplan 1989), which extend the CFG with formal descriptions of

[8]Grammar G is said to be in Chomsky normal form if all rules of G are of the form $A \rightarrow BC$ or $A \rightarrow a$, where A, B, and C are non-terminals and a is a terminal. Any context-free grammar can be converted to an equivalent Chomsky normal form.

grammatical units (words, phrases and sentences), so-called feature structures. The feature structures are sets of attribute-value pairs which can be nested, i.e. the values themselves can be atomic symbols or feature structures. The purpose of the features is to put some constraint on grammatical units — hence the name constraint-based model. These constraint-based models are also sometimes referred to as unification grammar because they "share a uniform operation for the merging and checking of grammatical information, which is commonly referred to as unification" (Uszkoreit and Zaenen 1995). According to Aït-Mokhtar et al. (2002), the main disadvantage of these systems is the lack of robustness.

Another class of popular full parsing methods is based on probabilistic context-free grammars (PCFG). These methods automatically extract grammar rules from a treebank (see Section 2.4.5) and ambiguity is resolved using probabilities estimated from the treebank. These methods assign a probability to every parse tree T (or at least to all parses it constructs) of a given sentence S and return the most likely parse: $P(S) = argmax_T P(T|S)$.

The disadvantage with probabilistic parsing methods is that a large quantity of annotated training data, i.e. a treebank, is needed. Moreover, the learnt rules and associated probabilities are very dependent on the treebank at hand. On the other hand, the main advantage is that the grammar rules are automatically extracted, but not written by hand.

Early probabilistic methods used conditional probability models for grammar rules based on POS tags and parent rules (e.g. Magerman and Marcus (1991)). Later probabilistic parsing methods have used lexical (word) information as well. The methods by Collins (1996) and Charniak (1997a) use lexical dependencies between heads of phrases to guide the parser[9]. Heads of phrases are used because of "the common linguistic intuition that the forms of a constituent and its sub-constituents are determined more by the constituent's head than any other of its lexical items" (Charniak 1997a).

Experience has shown that using lexical information does result in a higher parsing accuracy. A good English statistical parser which does not use lexical information, i.e. only probabilities computed from POS sequences and parent rules, can achieve about 75% labelled average precision/recall[10]. Contrastingly, a parser which, additionally, uses information on how par-

[9]The head of a phrase is the most important lexical item of the phrase. For example, in the noun phrase *a probabilistic method* the noun *method* is the head word.

[10]Refer to Section 2.4.6 for the definition of labelled precision and recall.

ticular English words fit together achieves labelled precision/recall rates of about 87-88% (Charniak 1997b).

Dependency parsing is one form of full parsing which has gained increasing interest in recent years. "One important reason seems to be that dependency parsing offers a good compromise between the conflicting demands of analysis depth, on the one hand, and robustness and efficiency, on the other" (Nivre and Scholz 2004). Dependency parsing does not generate constituents, but, rather, syntactic structure is indicated by relations, called dependencies, between the words of a sentence. Nivre and Hall (2005) have developed a language-independent system based on dependency parsing, which has been applied to several languages. Apparently, an advantage of this method is that a reasonable performance is obtained using a relatively small treebank (on the order of 100k words or less) for training. In other words, evaluation showed that the parsing accuracy of structurally similar languages was quite similar despite a large difference in the size of the training data for these languages. Note that, a treebank in dependency format is needed for the training phase (a conversion from phrase structures to dependency structures is possible).

2.4.2 Shallow parsing

The main problem with full parsing is that the set of solutions can grow exponentially because, generally, the parser considers all possible analyses of a given sentence. Moreover, since the goal is to build a complete parse tree for each sentence, the parser sometimes rejects correct analyses of a sentence part on lower levels in the parse tree on the ground that it does not fit into a global parse. Shallow parsing techniques do not have these problems because their aim is "to recover syntactic information efficiently and reliably from unrestricted text, by sacrificing completeness and depth of analysis" (Abney 1996). It is important to note that no attempt is made to resolve syntactic ambiguity in shallow parsers.

In many NLP applications, it can be sufficient to analyse sentence parts or phrases (e.g. noun phrases) without demanding that the particular parts fit into a global parse tree. This can be the case in areas like information extraction and text summarisation, where identification of phrases is more important than a global parse. Additionally, in cases of low quality input or spoken language, a shallow parsing method can be more robust[11] than a full

[11] Aït-Mokhtar et al. (2002) define robustness "as the ability for a language analyser

parsing method because of noise, missing words and mistakes in the input (Li and Roth 2001).

Abney's work on chunk parsing (Abney 1991) has been influential for recent work on shallow parsing. Abney argues that the human parser processes sentences by a chunk-by-chunk strategy and that there is psychological evidence for the existence of chunks, such as pause durations in reading. Accordingly, chunks correspond, in some way, to prosodic patterns. In later work, chunks have, additionally, been defined as non-recursive phrases. This penultimate sentence contains the following chunks:

[Accordingly] [chunks correspond] [in some way] [to prosodic patterns].

The parser implemented by Abney consists of a Chunker and an Attacher. The Chunker, which uses a simple CFG to describe the structure of chunks, converts a stream of word-tag pairs into a stream of chunks by using a bottom-up LR parsing technique (Aho et al. 2007). The Attacher attaches one chunk to another with the purpose of converting a stream of chunks into a stream of complete parse trees. Dividing the parsing process up between these two modules allows for simple CFG techniques in the Chunker and limits the use of more expensive techniques, for resolving attachment ambiguities, to the parts of the grammar where they are needed, i.e. in the Attacher.

2.4.2.1 Finite-state parsing techniques

Instead of using standard bottom-up or top-down parsing techniques which rely on CFGs, non-recursive language models, like finite-state grammars, have been used successfully to produce shallow parsers from the early 1990's.

The method by Koskenniemi (1990, 1992) was influenced by the CG approach, where, instead of using phrase tree structures to represent parses, syntactic tags are associated with words. First, each sentence to be analysed is fed through a morphological analyser which returns all morphological interpretations and all possible syntactic functions (see Section 2.4.3) of the analysed words. The sentences returned are represented as regular expressions, or equivalently as finite-state automata, which list all the combinatoric

to provide useful analysis of real-world input text, such as web pages, news, technical documentation, e-mail or FAQs."

possibilities to interpret them. Second, morphological disambiguation (tagging) is carried out. At last, sentences are run through finite-state rules to choose the correct type of boundaries between each two words and determine the correct syntactic tag for each word.

As input sentences, the rules are represented using regular expressions and transformed into finite-state networks using a compiler. "The whole finite-state grammar consists of a set of rules which constrain the possible choices of word interpretations, tags and boundaries to only those which are considered grammatical" (Koskenniemi et al. 1992). The purpose of the parser is to compute the intersection of the sentence automaton and each rule automaton. The result is then, preferably, one correct grammatical reading of the input sentence. It is noteworthy in this framework that the parsing does not build any new structures, but, rather, consists of excluding ungrammatical readings from a set of all possible readings returned by the morphological analyser.

The above method has been called a *reductionist* approach because all possible readings of a sentence are reduced to one correct reading by a set of elimination rules. A contrasting method is the *constructive* approach which is based on a lexical description of a collection of syntactic patterns. The use of finite-state transducers (also called finite-state markers (Grefenstette 1996)) to introduce syntactic labels into the input sentences is one example of the constructive approach.

At the Xerox Research Centre (XRC), extensions to the standard regular expression calculus have been developed, in order to create finite-state transducers for syntactic processing (Karttunen et al. 1996). Furthermore, XRC has developed a finite-state compiler (XFST), based on this calculus, for building and manipulating finite-state networks. In addition to the standard regular expression operators (union, concatenation, optionality, Kleene operators, complement, intersection, etc.), XFST supports special high level operators: restriction, replacement and longest match (Karttunen et al. 1996). The restriction operator is used to exclude unwanted analysis and the replacement operator to replace a string with another string with or without regard to a given context. The longest match operator is a special kind of replacement operator which only replaces the longest matched string.

A common constructive approach is to string together a sequence of transducers to build incremental (or cascading) shallow parsers (Grefenstette 1996, Abney 1997). Each transducer adds syntactic information into the text, such as brackets and names for grammatical functions. For example, in (Grefen-

stette 1996), a first transducer marks noun and verb phrases, a second transducer marks head words within groups (by using markers set by the previous transducer) and the last transducer applies syntactic filters to extract non-contiguous syntactic dependencies (like subjects of verbs), again using markers set by previous transducers.

Another (similar) constructive finite-state shallow parsing methods is based on Cascading Finite-State Automata (Abney 1997). It is called cascading because the parser uses a sequence of levels where a sentence part (phrase) on one level is built on a sentence part on a previous level. In other words, the parser consists of a sequence of finite-state automata, where each automaton contains a grammar part which is responsible for recognising a particular sentence part. In contrast to incremental finite-state transducers, where syntactic labels are inserted into the word stream, finite-state cascades reduce the input tokens to single elements, as in traditional bottom-up parsing. The grammar written for a particular level does not include recursion, i.e. phrases never contain same level or higher level phrases. The grammar for each level consists of a category and a regular expression which is translated to a finite-state automaton. The union of the automata yields a single, deterministic recogniser.

The hybrid method by Aït-Mokhtar and Chanod (1997) merges the constructive and the reductionist approaches by defining chunks (core phrases) by constraints rather than syntactic patterns.

Finite-state parsing methods have been used to develop a number of shallow parsers for different languages, e.g. Spanish (Molina et al. 1999), Swedish (Megyesi and Rydin 1999), German (Schiehlen 2003) and French (Chanod and Tapanainen 1996). The advantage of finite-state methods is that they are generally less domain-specific than probabilistic methods because large quantities of training data are not needed. Moreover, parsers built using finite-state methods are usually robust and fast because they are, in fact, just a pipeline of lexical analysers. On the other hand, finite-state methods are, by nature, not as complete as full parsing methods.

Aït-Mokhtar et al. (2002) have stated that "incrementality is a methodological principle commonly used to build robust, broad-coverage parsers that rely on computationally tractable syntactic descriptions". They argue that incremental parsing has two properties that distinguish it from more traditional parsing: *self-containment* and *descriptive decomposition*[12].

[12] Interestingly, Aït-Mokhtar et al. (2002) have used the idea of incrementality to build

Each incremental rule (transducer) is self-contained in the sense that its result depends only on the contextual restriction stated in the rule and the results accumulated by previous rules (transducers). If a substring matches a pattern stated in a rule, the corresponding result is generated without any backtracking — if no matches are found the rule does not have any affect on the results generated so far. This behaviour is different from traditional full parsing, in which rules can match substrings of the input (as in bottom-up parsing), but the parser might reject the analysis on the ground that it does not fit into a global parse.

The second item, descriptive decomposition, actually enforces modularity. Since each rule (transducer) is responsible for recognising a particular syntactic construction, the overall task of syntactic annotation is automatically divided into a number of subtasks. Broad coverage can thus be obtained by using a variety of subtasks, which as a whole are responsible for recognising a variety of linguistic constructions occurring in real texts.

2.4.3 Grammatical function analysis

Many parsing systems (both full and shallow) not only derive sentence structure but, additionally, analyse dependencies between the sentence's individual constituents. Different constituents can belong to the same category (e.g. noun phrase) but can have different grammatical functions (GFs). The purpose of GFs analysis is, for example, to identify the subjects and objects of verbs. Subcategorisation frames for verbs can be used to help attach the right arguments to the verbs, i.e. to distinguish between arguments and adjuncts of verbs. For example, in the sentence "*John ate the soup in the restaurant*", only the subject "*John*" and the object "*the soup*" are required arguments but the adjunct "*in the restaurant*" is optional.

Usually, grammatical functions and syntactic structures are annotated by two different components. "While shallow structures describe constituents which are subject to syntactic restrictions, GFs describe the relations between those constituents [...] while the order of tokens in a chunk is relatively fixed, the order of the GFs is relatively free ..." (Müller 2004). This generally means that syntactic structures can be annotated using POS tags only, while the annotation of GFs requires additional morphological information and a dictionary containing information about verb subcategorisation.

an *incremental deep* (full) parser for French.

Finite-state techniques have been used successfully to annotate GFs. Grefenstette (1996) uses "syntactic filters", which specify what must not appear between two items, to extract non-contiguous syntactic dependencies. These filters depend on the labels inserted by previous transducers, e.g. labels marking heads of noun phrases and types of verbs.

A GFs annotation for German also uses a cascade of finite-state transducers (Müller 2004). The main emphasis in that project is to make a distinction between complements (e.g. object-nominative, object-dative and object-accusative), typically subcategorised by a verb or an adjective. The component annotating the GFs uses all the linguistic information added by the previous shallow-structure-marking transducers.

2.4.4 Designing a parsing scheme

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

Voutilainen points out that the parsing 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 correctly annotate the sentences in the GDC.

2.4.4.1 The EAGLES guidelines

In 1996, EAGLES proposed guidelines for syntactic annotation of corpora (EAGLES, Expert Advisory Group for Language Engineering Standards 1996).

In the proposal, the following layers of information are recognised (which may or may not be encoded in a particular syntactic annotation scheme):

1. **Bracketing of segments**. Involves the delimitation of segments, usually with square brackets, which are recognised as having a syntactic integrity (sentences, clauses, phrases, words).

2. **Labelling of segments**. Indication of formal category of the constituents identified by the bracketing, such as a noun phrase, a verb phrase, a prepositional phrase, etc.

3. **Showing dependency relations**. Head-dependent relations between words, e.g. adjectives and the nouns they modify. Usually shown with dependency trees: a set of arrows pointing from a head to a dependent (or *vice verse*).

4. **Indicating functional relations**. Labelling of segments according to their syntactic function, such as subject, object, predicate, etc.

5. **Marking subclassification of syntactic segments**. Assigning feature values to phrases or words, e.g. marking a noun phrase as singular or a verb phrase as past tense.

6. **Deep or logical information**. This includes a variety of syntactic phenomena, such as co-referentiality, cross-reference and syntactic discontinuity.

7. **Information about the rank of a syntactic unit**. This is obtainable from most parser outputs by the embedding of marked brackets.

8. **Special syntactic characteristics of spoken language**. Indication of false starts, reiterations, pauses etc.

2.4.5 Treebanks

A treebank is a syntactically annotated corpus, in which the annotations follow a particular annotation scheme. Most treebanks have been built by manually, or semi-automatically, adding syntactic annotations to a POS tagged corpus. Treebanks have, for example, been used to facilitate linguistic research, as training corpora for data-driven methodologies and as evaluation resources for parsers. Three main kinds of annotation are used in practise: annotation of constituent structure, annotation of functional structure (grammatical functions) and theory-specific annotation (Nivre 2002).

The annotation found in most treebanks is, in fact, a combination of two or even three of theses categories.

The annotation of constituent structure (or bracketing) is the most common annotation method. It is, for example, used in the well known Penn Treebank (Marcus et al. 1993). Usually, this kind of annotation consists of POS tags for individual words, augmented with annotation of major constituents, like noun phrases, prepositional phrases, verb phrases, etc. These schemes are "usually intended to be theory-neutral and therefore try to use mostly uncontroversial categories that are recognised in all or most syntactic theories that assume some notion of constituent structure" (Nivre 2002). Thus, the advantage with this annotation method is that the treebank can be used by a larger group of researchers working within different theoretical frameworks. The disadvantage, however, is the risk that the annotation contains to little information which makes the treebank inadequate to use for anyone.

In recent years, annotation of functional structure has become increasingly important. First, grammatical function annotation has been added to many corpora annotated with constituent structure, e.g. the Penn Treebank II (Marcus et al. 1994). Secondly, so-called dependency syntax annotation schemes (signifying dependencies between words) have been developed, in which dependency structure is added directly on top of morphological information without any bracketing. The Prague Dependency Treebank of Czech is probably the best known example of this type of annotation structure (Hajič 1998).

The third kind of annotation scheme is the one which uses representations from a particular grammatical theory, for example HPSG, to annotate sentences. The advantage with theory-supporting treebanks is that they are more useful for people working with the selected type of grammatical theory, but the disadvantage is that they are not as appropriate for people who do not use the specific theoretical framework.

2.4.6 Evaluating parsers

A variety of parser evaluation methods have been used in the literature — for a general overview the reader is referred to Carroll et al. (1998). In this section, we only describe the most common method used — the *Parseval* scheme (Black et al. 1991, Grishman et al. 1992).

The original version of this scheme only compares phrase structure brackets in the output of a parser with the brackets in a treebank, i.e. in a *gold*

standard[13], without considering the constituent labels. Two main metrics are used, precision and recall, defined in the following manner:

$$precision = \frac{\#\text{ of correct brackets produced by parser}}{\text{total }\#\text{ of brackets produced by parser}} \tag{2.5}$$

$$recall = \frac{\#\text{ of correct brackets produced by parser}}{\#\text{ of brackets in treebank}} \tag{2.6}$$

These two metrics are then normally combined into the so-called F-measure, the weighted harmonic mean of precision and recall:

$$F_\alpha - measure = \frac{(1+\alpha)\cdot(precision \cdot recall)}{\alpha \cdot precision + recall} \tag{2.7}$$

When $\alpha = 1$, we get the widely used F_1-*measure* (in which precision and recall are evenly weighted):

$$F_1 - measure = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{2.8}$$

Another measure originating in *Parseval* is the mean number of *crossing brackets* per sentence. This measures the number of bracketed sequences in the parser which overlap with one from the treebank and where neither is properly contained in the other.

A more recent version of the original *Parseval* scheme takes labelling into account, i.e. the measures used are labelled precision and labelled recall (e.g. (Collins 1996)). Sekine and Collins (1997) have developed a widely used program, which automatically computes labelled precision, recall and F_1-measure.

Most work on evaluation for full parsing has been focused on the English language using the *Parseval* scheme and the Penn Treebank as the gold standard. Because of this dominant paradigm, it has been pointed out that parser development might be tuned in order to obtain high precision and recall scores when tested using *Parseval* and the Penn Treebank (Gaizauskas et al. 1998). In order to filter out the particular grammar formalism implicit in the Penn Treebank, Gaizauskas et al. advocate using a gold standard that "encodes for each sentence only the constituents upon which there is

[13] A gold standard is a benchmark which is widely accepted as being correct or the best available.

broad agreement across a range of grammatical theories", basically resulting in flatter structure.

Another path of work with regard to evaluation for full parsing in recent years is dependency-based evaluation (Lin 1998). Instead of comparing phrase boundaries, comparisons are made between the dependency relations in the parser output and the gold standard.

In the case of shallow parsing, the combination of *Parseval* and a gold standard with full parsers (e.g. Penn Treebank) is not suitable because obviously recall will be very low. If a partially parsed treebank is not available, a dependency based evaluation might be more appropriate (Kübler and Heike 2002).

2.4.7 Parsing Icelandic text

To our knowledge, only one parser has yet been developed for parsing Icelandic text. The project, which concluded in 2005, was carried out at the Icelandic software company Frisk Software International. In November 2004, 36 man months had been spent on the project (Albertsdóttir and Stefánsson 2004). The parser is a full parser based on HPSG and uses a proprietary morphological analyser developed by the same company. The purpose of the project is to build a basic NLP unit which can be used for further development of NLP tools.

One other project regarding syntactic analysis of the Icelandic language is, indeed, worth mentioning. A corpus of spoken Icelandic (ÍS-TAL) has been compiled and POS tagged using the *TnT* tagger and the large Icelandic tagset (Rögnvaldsson 2005). Moreover, in this project, scripts have been developed that convert morphological information into syntactic information. This conversion is possible due to the substantial amount of syntactic information included in the Icelandic POS tags. For example, nouns, pronouns and adjectives are marked by case which should make it possible to deduce subject or object function for these words. Additionally, words can be grouped into phrases by using the fact that feature agreement between a noun and its modifiers is a characteristic of Icelandic noun phrases. This syntactic annotation is then enhanced by using a few simple hand-written rules for correcting the syntactic tags according to the syntactic environment and assigning some hierarchical structure to the sentences. The result is "a rather flat structure with lots of errors and inconsistencies [...] but it is far better than having no syntactic information at all" (Rögnvaldsson 2005).

Chapter 3

The Icelandic Language

The Icelandic language is one of the Nordic languages which comprise the North-Germanic branch (Danish, Swedish, Norwegian, Icelandic, Faroese) of the Germanic language tree. Linguistically, Icelandic is most closely related to Faroese and the dialects of Western Norway. Iceland was mainly settled in the late ninth and early tenth century, mostly by people from Western Norway, but settlers also came from the British Isles. The language which prevailed was that of the people from Western Norway.

The Icelandic language is a morphologically rich language, due to inflectional complexity. From a syntactic point of view, Icelandic has a basic subject-verb-object (SVO) word order, but, in fact, the word order is relatively flexible, because morphological endings carry a substantial amount of syntactic information.

In this chapter we will, first, in section 3.1, briefly describe the morphology of the Icelandic language, and then, briefly, discuss syntactic issues in section 3.2.

3.1 Morphology

Icelandic words are usually divided into 3 categories (11 word classes): nominals (nouns, adjectives, pronouns, the definite article and numerals), verbs and non-inflected words (conjunctions, prepositions, adverbs (some of which, in fact, are inflected for comparison), the infinitival marker and interjections). Let us now briefly consider the different word classes — for a thorough description of the Icelandic language the reader is referred to (Þráinsson 1994,

Svavarsdóttir 2005).

3.1.1 Nominals

The main feature of nominals is declension. Icelandic has four cases: nominative, accusative, dative and genitive. The oblique cases, accusative, dative and genitive, are mainly controlled by verbs and prepositions. Additionally, most nominals are marked by number (singular and plural) and gender. There are three genders: masculine, feminine and neuter.

3.1.1.1 Nouns

"The gender for nouns is primarily a formal characteristic, and even if the grammatical gender generally matches the natural gender or sex in words denoting animates, it is still highly arbitrary [...]" (Svavarsdóttir 2005).

Generating the plural for Icelandic nouns is more complicated than in the other Nordic languages, as well as in English. In contrast to English, Icelandic does not have a general rule, which applies to all the genders, for generating the plural. There are specific endings for plural masculine nouns, other endings for plural feminine nouns and, usually, no specific endings for plural neuter nouns. Moreover, vowel mutation often occurs in the plural, especially for neuter nouns.

Icelandic nouns are often divided into a number of inflectional classes (see (Þráinsson 1994) for a thorough discussion). The classification is based on the ending of the genitive singular, the gender, and the ending of the nominative plural.

Table 3.1 shows an example of the declension for nouns belonging to three different inflectional classes: the (indefinite) masculine noun "*fiskur*" (fish), the (indefinite) feminine noun "*tunga*" (tongue), and the (indefinite) neuter noun "*borð*" (table). The common part shared by all forms, the stem, is "*fisk*", "*tung*", and "*borð*".

Nouns can have a suffixed article which makes them definite (see below).

3.1.1.2 Adjectives

Adjectives have a strong or a weak declension. A strong declension is used in indefinite noun phrases whereas a weak declension is used in definite noun phrases. Moreover, adjectives inflect in the positive and in the superlative.

Case	Singular	Plural	Singular	Plural	Singular	Plural
Nominative	fisk-ur	fisk-ar	tung-a	tung-ur	borð	borð
Accusative	fisk	fisk-a	tung-u	tung-ur	borð	borð
Dative	fisk-i	fisk-um	tung-u	tung-um	borð-i	borð-um
Genitive	fisk-s	fisk-a	tung-u	tung-na	borð-s	borð-a

Table 3.1: Examples of the declension for Icelandic nouns

Tables 3.2 and 3.3 show an example of a strong and a weak declension, respectively, for the adjective "*rauður*" (red). "*rauður fiskur*" stands for the indefinite masculine *a red fish* (strong declension), while "*rauði fiskurinn*" stands for the definite *the red fish* (weak declension). In the weak declension, notice the masculine definite article "*inn*" as an inflectional ending for the noun, e.g. "*rauði fiskur-inn*".

3.1.1.3 Pronouns

Pronouns are divided into 6 subcategories; demonstrative pronouns, reflexive pronouns, possessive pronouns, indefinite pronouns, personal pronouns and interrogative pronouns. An example of each category is shown in table 3.4. Table 3.5 shows the inflection of the first person personal pronoun, "*ég*" (I) in the four cases, singular and plural.

3.1.1.4 Article

The Icelandic definite article is only one word, i.e. "*hinn*". The article, which exists in the three genders, singular and plural, makes the nouns definite. Suffix use of the article is the most common and in that case the article is not considered a special word, but rather an inflectional ending. Table 3.6 shows the declension of the singular words, "*maðurinn*", "*konan*" and "*barnið*" (the man (masculine), the woman (feminine), the child (neuter)).

3.1.1.5 Numerals

Icelandic numerals are ordinals or cardinals. The ordinals inflect like nouns as well as the first four cardinals: "*einn*", "*tveir*", "*þrír*", "*fjórir*" (one, two, three, four). Table 3.7 shows the declension of the cardinal "*einn*".

	Strong declension	
Case	Singular	Plural
Nominative	rauð-ur fisk-ur	rauð-ir fisk-ar
Accusative	rauð-an fisk	rauð-a fisk-a
Dative	rauð-um fisk-i	rauð-um fisk-um
Genitive	rauð-s fisk-s	rauð-ra fisk-a

Table 3.2: An example of a strong declension for the adjective "*rauður*" (modifying the noun "*fiskur*")

	Weak declension	
Case	Singular	Plural
Nominative	rauð-i fisk-ur-inn	rauð-u fisk-ar-nir
Accusative	rauð-a fisk-inn	rauð-u fisk-a-na
Dative	rauð-a fisk-i-num	rauð-u fisk-u-num
Genitive	rauð-a fisks-ins	rauð-u fisk-a-nna

Table 3.3: An example of a weak declension for the adjective "*rauður*" (modifying the noun "*fiskur*")

Category	Example	English equivalent
Demonstrative	sá, þessi, hinn	this, that, the other one
Reflexive	sig	himself/herself/itself
Possessive	minn, þinn, sinn	mine, yours, his
Indefinite	annar, enginn, allir	other, nobody, all
Personal	ég, þú, hann/hún/það	I, you, he/she/it
Interrogative	hver, hvor	who, which

Table 3.4: Examples of Icelandic pronouns

Case	Singular	Plural
Nominative	ég	við
Accusative	mig	okkur
Dative	mér	okkur
Genitive	mín	okkar

Table 3.5: Inflection of the first personal pronoun *ég* (I)

Case	Masculine	Feminine	Neuter
Nominative	maður-inn	kona-n	barn-ið
Accusative	mann-inn	konu-na	barn-ið
Dative	manni-num	konu-nni	barni-nu
Genitive	manns-ins	konu-nnar	barns-ins

Table 3.6: Examples of suffixed article declensions

Case	Masculine	Feminine	Neuter
Nominative	ein-n	ein	eitt
Accusative	ein-n	ein-a	eitt
Dative	ein-um	ein-ni	ein-u
Genitive	ein-s	ein-nar	ein-s

Table 3.7: Declension of the cardinal “einn”

Person	Present tense		Past tense	
(singular/plural)	Singular	Plural	Singular	Plural
1^{st} (ég/við)	borð-a	borð-um	borð-aði	borð-uðum
2^{nd} (þú/þið)	borð-ar	borð-ið	borð-aðir	borð-uðuð
3^{rd} (hann/þeir)	borð-ar	borð-a	borð-aði	borð-uðu

Table 3.8: Conjugation of a regular verb in indicative mood and active voice

Person	Present tense		Past tense	
(singular/plural)	Singular	Plural	Singular	Plural
1^{st} (ég/við)	borð-i	borð-um	borð-aði	borð-uðum
2^{nd} (þú/þið)	borð-ir	borð-ið	borð-aðir	borð-uðuð
3^{rd} (hann/þeir)	borð-i	borð-i	borð-aði	borð-uðu

Table 3.9: Conjugation of a regular verb in subjunctive mood and active voice

3.1.2 Verbs

Icelandic verbs conjugate in person, number, tense, mood and voice. There are two morphological tenses: present tense and past tense. A finite verb form can have any of the three moods: indicative, subjunctive or imperative. Verbs can also have three infinite forms, i.e. the infinitive form and two participles: present and past. The voices are active and passive. Table 3.8 shows the conjugation of the regular verb "*borða*" (eat) for the present and past tenses in indicative mood and active voice. The subjunctive form of the same verb appears in table 3.9. Note that for regular verbs the past tense is the same in indicative and subjunctive moods.

Ablaut appears in the stem of irregular verbs when they are conjugated. Table 3.10 shows the irregular verb "*bíta*" (bite), in indicative mood. The subjunctive form of the same verb appears in table 3.11.

Past and present participle is used with the copula "*vera*" (be), e.g. "*hann var bitinn*" (he was bitten) and "*ég er hlaupandi*" (I am running). The present participle is easily recognised by the suffix "*-andi*". The infinitive is "*að*" (to), i.e. "*að lesa*" (to read), "*að bíta*" (to bite), etc.

In Icelandic, transitive verbs govern cases. As an example, consider the

Person	Present tense		Past tense	
(singular/plural)	Singular	Plural	Singular	Plural
1^{st} (ég/við)	bít	bít-um	beit	bit-um
2^{nd} (þú/þið)	bít-ur	bít-ið	bei-st	bit-uð
3^{rd} (hann/þeir)	bít-ur	bít-a	beit	bit-u

Table 3.10: Conjugation of an irregular verb in indicative mood and active voice

Person	Present tense		Past tense	
(singular/plural)	Singular	Plural	Singular	Plural
1^{st} (ég/við)	bít-i	bít-um	bit-i	bit-um
2^{nd} (þú/þið)	bít-ir	bít-ið	bit-ir	bit-uð
3^{rd} (hann/þeir)	bít-i	bít-i	bit-i	bit-u

Table 3.11: Conjugation of an irregular verb in subjunctive mood and active voice

sentence "*Ég borða fisk*" (I eat fish). The nominative case for fish is "*fiskur*" but here the verb "*borða*" demands an accusative case object. As another example, consider the verb "*vera*" (be) which demands a predicate nominative. In the sentence "*Þetta er fiskur*" (This is a fish), the noun "*fiskur*" is in the nominative case.

3.1.3 Non-inflected words

The non-inflected word classes in Icelandic are conjunctions, prepositions, adverbs, the infinitival marker and interjections. Excluding adverbs, these classes are closed, i.e. new words do not enter them.

3.1.3.1 Conjunctions

Conjunctions in Icelandic are of two kinds; coordinating conjunctions and subordinate conjunctions. The predominant coordinating conjunctions are: "*og*" (and), "*en*" (but) and "*eða*" (or). The main subordinate conjunctions are: "*að*" (that), "*sem*" (which), "*ef*" (if) and "*þegar*" (when).

3.1.3.2 Prepositions

Prepositions govern oblique cases in Icelandic. The following is a list of some of the prepositions that govern the accusative, dative or genitive cases.
Accusative: "*um*" (about), "*í*" (in), "*á*" (on), "*yfir*" (over), "*undir*" (under), "*með*" (with), "*eftir*" (after), "*við*" (by) , "*fyrir*" (for), "*gegnum*" (through), "*kringum*" (around), "*upp*" (up), "*niður*" (down).
Dative: "*frá*" (from), "*af*" (of), "*undan*" (from under), "*að*" (towards), "*samkvæmt*" (according to), "*ásamt*" (together with), "*handa*" (for), "*gegn*" (against), "*meðfram*" (along).
Genitive: "*til*" (to), "*án*" (without), "*auk*" (in addition to), "*meðal*" (among), "*milli*" (between), "*vegna*" (because of).

Note that some of the prepositions (e.g. "*í*", "*á*", "*undir*", and "*yfir*") can govern both the accusative and the dative case.

3.1.3.3 Adverbs

Adverbs are the biggest category of the non-inflectional words and the only one which accepts new words, i.e. adverbs that can be derived from adjectives. In some cases an adverb takes on the role of a preposition when it governs the case of a nominal. A few adverbs can compare. Adverbs are usually subcategorized, like "*háttaratviksorð*" (adverbs of manner), "*tíðaratviksorð*" (adverbs of time and frequency), "*staðaratviksorð*" (adverbs of place), "*áherslu-atviksorð*" (adverbs of degree), and "*spurnaratviksorð*" (adverbs of question).

3.1.3.4 The infinitive

Only one word is in this category, i.e. "*að*" (to). "*Að*" is used with a verb in the infinitive mood, e.g. "*að borða*" (to eat).

3.1.3.5 Interjections

Interjections are words that are "shouted" and describe, for example, joy, astonishment or fear. An example of words in this category is: "*hæ*" (hi), "*hó*" (whoopee), "*æ*" (ouch), "*já*" (yeah), "*nei*" (nah), "*uss*" (shush).

3.2 Syntax

The following passage, borrowed from Svavarsdóttir (2005), is a good introduction to a discussion on Icelandic syntax:

> Inflection takes much of the load of showing the internal structure of sentences. Agreement in gender, case and number between a noun and its modifiers (adjectives, pronouns, etc.) indicates which words belong together and form constituents, and case marking differentiates them and is indicative of their syntactic function. Subjects are, for example, typically in the nominative and most objects in either accusative or dative, depending on the verb. Furthermore, the finite verb agrees with the subject in person and number. Word order can thus be relatively free, even if the freedom is restricted in many ways. Icelandic has a basic SVO order (subject – verb – object), in subordinate as well as in main clauses, and any deviations from that are syntactically, semantically or stylistically marked.

In the following sections, we discuss phrases (constituents), grammatical (syntactic) functions and word order in more detail.

3.2.1 Phrases

In a discussion on syntax, it is, generally, assumed that sentences are made out of group of words, called phrases. Each phrase contains a *head* (the main word) which determines the type of the phrase. The typical phrase consists of the head, an optional modifier (specifier) preceding the head, and an optional complement following the head. Both the modifier and the complement can be phrases themselves. In Icelandic, nominal modifiers agree with the head in gender, number and case. In contrast, the head, generally, governs the case of its nominal complements.

Phrases in Icelandic are, broadly speaking, categorised into the familiar five type of phrases: adverb phrase ($AdvP$), adjective phrase (AP), noun phrase (NP), preposition phrase (PP) and verb phrase (VP).

An AdvP consists of a sequence of one or more adverbs, of which the rightmost adverb is the head. Similarly, an AP contains a sequence of one or more adjectives. Additionally, an AP includes an optional AdvP as a modifier. Clause 3.1 shows the phrase "*mjög góður*" (very good), an example

of an AP along with a modifying AdvP (we use brackets and labels to indicate the beginning and end of a phrase):

[AP [AdvP mjög AdvP] góður AP] (3.1)

A NP comprises a sequence of optional modifiers followed by a noun (or a pronoun) as the head. The order of the modifiers is strict, i.e. indefinite pronoun, demonstrative pronoun, numeral and adjective (phrase). A typical example, "*Allir þessir fjóru fallegu hestar*" (All these four beautiful horses) is given in 3.2.

[NP Allir þessir fjórir [AP fallegu AP] hestar NP] (3.2)

Agreement, in gender, number and case between a noun and its modifiers, is a characteristic of Icelandic noun phrases. The modifiers "inherit" their feature values from the head word of the phrase. For example, in 3.2, the indefinite pronoun "*allir*", the demonstrative pronoun "*þessir*", the numeral "*fjórir*", the adjective "*fallegu*", and the noun "*hestar*" all appear in masculine, plural and nominative. As another example, in the sentence "*Öllum litlu börnunum var hjálpað*" (All (the) little children were helped) the indefinite pronoun "*öllum*", the adjective "*litlu*", and the noun "*börnunum*" all appear in the neuter, plural and dative.

The genitive complement, expressing possession, is an example of a noun complement phrase. Consider, for example, the clause "*þessir fallegu hestar litlu strákanna*" (these beautiful horses small boys), shown in 3.3.

[NP þessir [AP fallegu AP] hestar NP] [NP [AP litlu AP] strákanna NP] (3.3)

The phrase "*litlu strákanna*" appears in the genitive case and specifies who owns the horses.

A PP, typically, consists of an optional AdvP, a preposition (the head) and a following noun phrase as a complement. The preposition governs the case of the complement. For example, in the sentence "*Jón stóð við vegginn*" (John stood by (the) wall), shown in 3.4, the preposition "*við*" demands an accusative case for the noun "*vegginn*".

[NP Jón NP] [VP stóð] [PP við [NP vegginn NP] PP] (3.4)

A VP consists of a head verb followed by an optional object/complement

phrase, depending on the type of the verb.

3.2.2 Grammatical functions

In our discussion on grammatical functions, we will assume that phrases can have different functions, of which the most important are:

- A subject: a NP
- An object: a NP
- A (verb) complement: a NP/AP, or a past participle verb phrase

In declarative sentences, the subject is the phrase which precedes the (finite) VP and the object/complement the phrase which follows the VP.

In Icelandic, a subject is normally in the nominative case. The finite verb agrees with the subject in person and number. In the sentence "*Jón las bókina*" (John read (the) book) in 3.5, the finite verb "*las*" agrees with the nominative subject "*Jón*" in person (3^{rd}) and number (singular).

[NP Jón NP] [VP las [NP bókina NP] VP] (3.5)

Non-nominative subjects (often referred to as *quirky case*) are, however, not uncommon, e.g. "*mig (accusative) vantar*" (I need), "*mér (dative) leiðist*" (I'm bored), and "there is, thus, no one-to-one relationship between morphological case marking and grammatical function" (Þráinsson 1994). In the case of non-nominative subjects the verb is always in the 3^{rd} person (instead of agreeing with the subject in person).

A transitive verb govern the case (accusative, dative or genitive) of its object phrase, while an intransitive verbs either does not require an object or demands a nominative complement. In 3.5, the verb "*las*" demands an accusative object.

In the case of di-transitive verbs, generally, the first object (the indirect object) is in the dative case, whereas the second object (the direct object) is in the accusative — see for example the sentence "*Jón gaf Maríu bókina*" (John gave Mary (the) book) shown in 3.6:

[NP Jón NP] [VP gaf] [NP Maríu (dat.) NP] [NP bókina (acc.) NP] (3.6)

Other case patterns for the indirect and direct objects do indeed appear, i.e. accusative-dative, dative-dative, accusative-genitive and accusative-accusative.

In the sentence "*Jón er góður kennari*" (John is good teacher), the copula verb "*er*" (be) is an example of a verb which demands a nominative complement; see 3.7:

[NP Jón NP] [VP er [NP [AP góður AP] kennari NP] VP] (3.7)

3.2.3 Word order

Since the case marking gives an indication of the grammatical function, the subject-object word order in Icelandic is relatively free. Consider the following two sentences "*Jón borðaði fiskinn*" (John ate (the) fish) and "*Fiskinn borðaði Jón*" ((The) fish ate John; meaning: the fish, John ate), shown in 3.8 and 3.9:

[NP Jón (nom.) NP] [VP borðaði] [NP fiskinn (acc.) NP] (3.8)

[NP Fiskinn (acc.) NP] [VP borðaði] [NP Jón (nom.) NP] (3.9)

The difference in semantics between these two sentences in subtle. The latter puts emphasis on the fact the John ate the fish as opposed to John eating something else.

In Icelandic declarative clauses, the finite verb occupies the familiar Germanic verb-second (V2) positions. "Thus if something is preposed or topicalised the subject will follow the finite verb rather than precede it" (Þráinsson 1994) — see for example the sentences "*Oft las Jón bókina*" (Often read John (the) book) and "*Hverjum gaf Jón bókina?*" (To whom gave John (the) book?) shown in 3.10 and 3.11:

[AdvP Oft AdvP] [VP las] [NP Jón NP] [NP bókina NP] (3.10)

[NP Hverjum NP] [VP gaf] [NP Jón NP] [NP bókina NP]? (3.11)

Icelandic auxiliary or modal verbs precede the main verb, as exemplified

in the sentences "*Hann hafði farið burt*" (He had gone away), and "*Hún gat ekki borðað fiskinn*" (She could not eat fish-the), shown in 3.12 and 3.13:

[NP Hann NP] [VP hafði farið VP] [AdvP burt AdvP] (3.12)

[NP Hún NP] [VP gat [AdvP ekki AdvP] borðað VP] [NP fiskinn NP] (3.13)

Part II

Data and System

Chapter 4

Data

4.1 The corpora

4.1.1 The IFD corpus

The main corpus (and the tagset; see section 4.2) used in our research was created during the making of the Icelandic Frequency Dictionary (*IFD*) (Pind et al. 1991). The following description is borrowed from Helgadóttir (2004):

> The *IFD* corpus is considered to be a carefully balanced corpus consisting of about 590k tokens. All the 100 fragments of texts in the corpus were published for the first time in 1980–1989. The corpus comprises five categories of texts, i.e. Icelandic fiction, translated fiction, biographies and memoirs, non-fiction and books for children and youngsters. No two texts are attributed to the same person and all texts start and finish with a complete sentence.

Table 4.1 lists some statistics from the *IFD* corpus. In the table, *tokens* refers to word tokens, and *types* refer to word types, i.e. different words.

It is interesting that the ratio of ambiguous tokens, 59.66%, is much higher than for languages like English (38.65%), Czech (45.97%), Estonian, (40.24%), Hungarian (21.58%), Romanian (40.00%), Slovene (38.01%) (Hajič 2000). This seems to indicate that the most frequent tokens in Icelandic are (very) ambiguous. Table 4.2, which lists the 10 most frequent tokens along

with their *tag profile*[1], shows, indeed, that this conjecture is true (except for the first three tokens).

The computer files for the *IFD* corpus each contain one text excerpt. To make the corpus suitable for ten-fold cross-validation[2], the computer files of the corpus have been divided (by Helgadóttir) in the following way. Each file has been divided into ten approximately equal parts, and from these, ten different disjoint pairs of files have been created. Each pair consists of a training set, containing about 90% of the tokens from the corpus, and a test set, containing about 10% of the tokens. By using this procedure, each stratified set should contain a representative sample from all genres in the corpus. The test corpora are independent of each other, whereas the training corpora overlap and share about 80% of the examples. All words in the texts, except proper nouns, start with a lower case letter. Table 4.3 shows statistics for the 10 test corpora. A word in a test corpus is considered unknown if it does not appear in the associated training corpus.

4.1.2 Other corpora

In this thesis, we will, additionally, use the following five different tagged text segments for testing taggers. The first one consists of "young" literary works from the period after 1980. The second segment consists of "old" literary works from the last part of the 19th century and first part of the 20th century. The third segment consists of a text about computers and information technology acquired from the newspaper *Morgunblaðið*, a newsletter from the University of Iceland Computing Services and the web sites of several information technology companies. The fourth segment comprises text about law and business taken from various sources. The last segment consists of newspaper text from January 2004, acquired from *Morgunblaðið*.

The first four text segments are the same as were used in (Helgadóttir 2004), but with the following differences. First, we have removed texts from "young", which also appear in the *IFD* corpus. Secondly, we have hand-corrected various tagging errors that existed in these segments.

[1] In this thesis, we refer to the set of possible tags for a given word w as the *tag profile* for w.

[2] In ten-fold cross-validation a corpus is randomly divided into ten disjoint subsets of (approximately) equal size. The training is performed ten times on nine of these ten disjoint data sets and then testing is performed on the one left out, each time leaving out a different one.

Feature	Number	Ratio
Sentences	36,922	15.99[a]
Tokens	590,297	
Ambiguous tokens	352,200	59.66%
Ambiguity rate[b]	2.74	
Types	59,358	
Types occurring once	34,979	58.93%
Unambiguous types	49,995	84.16%
Ambiguous types	9,403	15.84%
Types occurring with 2 tags	6,568	11.07%
Types occurring with 3 tags	1,758	2.96%
Types occurring with 4 tags	577	0.97%
Types occurring with 5 tags	209	0.35%
Types occurring with 6 tags	92	0.15%
Types occurring with 7 tags	67	0.11%
Types occurring with 8 tags	26	0.04%
Types occurring with 9 tags	21	0.04%
Types occurring with 10 tags	18	0.03%
Types occurring with 11 tags	23	0.04%
Types occurring with 12 tags	7	0.01%
Types occurring with 13 tags	10	0.02%
Types occurring with 14 tags	9	0.02%
Types occurring with 15 tags	5	0.01%
Types occurring with 16 tags	1	0.00%
Types occurring with 17 tags	6	0.01%
Types occurring with 18-24 tags	6	0.01%

Table 4.1: Statistics for the *IFD* corpus

[a] Average sentence length.
[b] For all tokens.

Frequency	Token	English translation[b]	Tags[a]
33181	.		.
22176	og	and	c
22083	,		,
21011	að	to	cn_c_aþ_aa
15319	í	in	aþ_ao_aa
12450	á	on	aþ_ao_sfg1en_sfg3en_aa_nven_nveo_nveþ_au
8040	hann	he	fpken_fpkeo
7905	var	was	sfg3eþ_sfg1eþ_lkensf
7676	sem	that	ct_c_aa_sfg1en
6357	er	is	sfg3en_sfg1en_ct_c

Table 4.2: The 10 most frequent tokens in the *IFD* corpus

[a] "_" is used as a separator between tags. The tagset is explained in section 4.2.
[b] Corresponding to the most frequent tag.

Test corpus	# of tokens	# of sentences	unknown word ratio
01	59,169	3,503	7.57%
02	58,967	3,601	6.79%
03	59,077	3,541	6.88%
04	59,067	3,776	6.69%
05	59,075	3,861	6.51%
06	59,136	3,748	6.70%
07	59,109	3,688	6.70%
08	58,981	3,698	6.93%
09	59,143	3,743	6.80%
10	58,573	3,753	6.83%
Average:	59,030	3,691	6.84%

Table 4.3: Statistics for the ten test corpora derived from the *IFD* corpus

Test corpus	# of tokens	# of sentences	unknown word ratio
young	3,881	234	7.21%
old	6,023	226	8.88%
law/business	2,778	134	13.97%
computers	2,926	142	15.07%
newspaper	10,016	500	11.08%

Table 4.4: Statistics for the other corpora

We constructed a tagged version of the newspaper texts in the following manner. First, we used a combined tagger (see section 7.4) to perform initial tagging. Then, we processed the resulting file word by word and hand-corrected the tagging errors.

Table 4.4 shows statistics for these text segments. Here, a word in a test corpus is considered unknown if it does not appear in the *IFD* corpus.

4.2 The tagset

Due to the morphological richness of the Icelandic language, the main tagset (the *IFD* tagset) is large and makes fine distinctions compared to related languages. The rich inflections of an Icelandic word contribute more information about POS of surrounding words than is the case, for example, for English where word order is not as free. The tagset consists of 662 possible tags: 192 noun tags, 163 pronoun tags, 144 adjective tags, 82 verb tags, 27 numeral tags, 24 article tags, 16 punctuation tags, 9 adverb/preposition tags, 3 conjunction tags and 1 tag for foreign words and words not analysed. 639 tags of the possible 662 tags in the Icelandic tagset appear in the *IFD* corpus.

We can illustrate the preciseness of the tags by examining the semantics of a tag. Each character in a tag has a particular function. The first character denotes the word class. For each word class there is a predefined number of additional characters (at most six) which describe morphological features, like *gender*, *number* and *case* for nouns; *degree* and *declension* for adjectives; *voice*, *mood* and *tense* for verbs, etc.

Tables 4.5 and 4.6 show the semantics of the tags for nouns and adjectives,

Char #	Category/ Feature	Symbol – semantics
1	Word class	**n**–noun, **l**–adjective
2	Gender	**k**–masculine, **v**–feminine, **h**–neuter, **x**–unspecified
3	Number	**e**–singular, **f**–plural
4	Case	**n**–nominative, **o**–accusative, **þ**–dative, **e**–genitive
5	Article	**g**–with suffixed article
5	Declension	**v**–strong, **s**–weak
6	Proper noun	**m**–person, **ö**–place, **s**–other
6	Degree	**f**–positive, **m**–comparative, **e**–superlative

Table 4.5: The semantics of the tags for nouns and adjectives

Char #	Category/ Feature	Symbol – semantics
1	Word class	**s**–verb (except for past participle)
2	Mood	**n**–infinitive, **b**–imperative, **f**–indicative, **v**–subjunctive, **s**–supine, **l**–present participle
3	Voice	**g**–active, **m**–middle
4	Person	**1**–1^{st} person, **2**–2^{nd} person, **3**–3^{rd} person,
5	Number	**e**–singular, **f**–plural
6	Tense	**n**–present, **þ**–past

Table 4.6: The semantics of the tags for verbs

and the semantics of the tags for verbs, respectively. Consider, for example, the tag "*nken*". The first letter, "*n*", denotes the word class "*nafnorð*" (noun), the second letter, "*k*", denotes the gender "*karlkyn*" (masculine), the third letter, "*e*", denotes the number "*eintala*" (singular) and the last letter, "*n*", denotes the case "*nefnifall*" (nominative case).

To give another example, consider the phrase "*fallegu hestarnir stukku*" (the beautiful horses jumped). The corresponding tag for "*fallegu*" is "*lkenvf*", denoting adjective, masculine, singular, nominative, weak declension, positive; the tag for "*hestarnir*" is "*nkfng*", denoting noun, masculine, plural, nominative with suffixed definite article, and the tag for "*stukku*" is "*sfg3fþ*", denoting verb, indicative mood, active voice, 3-rd person, plural and past tense. Note the agreement in gender, number and case between the adjective and the noun, and the agreement in number between the subject and the

verb (the verb also agrees with the subject in person, but since all nouns are 3^{rd} person by default, the person feature is not overtly expressed in this case).

A complete description of the tagset can be found in the Appendix B.

Chapter 5

The Tagging System

The tagging system consists of a tokeniser/sentence segmentiser, a morphological analyser, *IceMorphy*, a linguistic rule-based tagger, *IceTagger* and a trigram tagger, *TriTagger*. The tokeniser is used for tokenising a stream of characters into linguistic units, *IceMorphy* for guessing the tags of unknown words, and *IceTagger* and *TriTagger* for assigning unambiguous tags to words in text.

Similar to a CG system, our tagging system is linguistic rule-based (excluding *TriTagger*). It, however, differs from a typical CG system in mainly four ways:

- Its main dictionary is automatically derived from the *IFD* corpus (the dictionary contains about 60,000 word forms), and is thus far from extensive. This means that *IceMorphy* mainly uses information derived from the *IFD* corpus. Moreover, the main dictionary has a number of *tag profile gaps*, as discussed in Section 5.2.5.

- As discussed in section 5.3.4, the number of constraints (local rules) are only about 175, instead of in the thousands as is common in CG taggers. In addition, the tagger uses general heuristics, discussed in section 5.3.5, as an aid in the disambiguation phase.

- In addition to POS tagging, typical CG systems also tag syntactic functions.

- The CG framework is designed to be language independent, but minimal effort is made in our system to support language independence.

The development time of our tagging system (excluding *TriTagger*) was only 7 man months which can be considered a short development time for a linguistic rule-based system. The short development time is mainly due to the emphasis on using heuristics for disambiguation, instead of writing a large number of rules. Furthermore, the dictionaries used by the system are (in most cases) automatically derived from the *IFD* corpus. We are not aware of other systems like this one, developed in such a short time frame, that outperform state-of-the-art data-driven methods which use supervised learning.

In this chapter, we describe each component of the tagging system in detail.

5.1 Tokenisation

Our tokeniser is used for tokenising input files and converting between different file formats.

The tokeniser reads an input file which can have three possible formats: one token per line, one sentence per line, or an unspecified format. First, if the input has one token per line (i.e. it has already been preprocessed) the main task of the tokeniser is sentence segmentation. Secondly, assuming the input consists of one sentence per line, the task of the tokeniser is to split each line up to linguistic units (tokens). Third, if the input format is unspecified (it could for example be a mixture of one token, many tokens and one sentence per line) the purpose of the tokeniser is, first, to perform sentence segmentation and, then, to perform tokenisation. The tokeniser can read input files in one format and produce output files in another format.

The main purpose of the sentence segmentiser is to "decide" when a token constitutes a full stop (end of an sentence). A sentence is constructed by reading the input file character by character and constructing a character string, which is returned to the calling program when a full stop is encountered. The candidates for a full stop are the tokens *. ! ? : "*. These characters are considered full stops if they are the last character of the current line or a space follows and various special conditions do not hold. Special conditions occur, for example, when the character is a period which is a part of a known abbreviation or a part of an ordinal number (e.g. 72. street).

Once sentence segmentation has been carried out, each sentence is split up into tokens. The task is, thus, to "decide" where one token ends and another

token starts. Each sentence is read character by character and substrings of the sentence are matched against various patterns. For example, a word starts with a letter and can be followed by any letter or digit or a special character from a predefined set (e.g. the characters "-" and "_"). One of the complications in producing tokens has to do with the period character, i.e. it needs to be determined when a period is part of a token. This is similar to the decision made when encountering a period during sentence segmentation.

5.2 IceMorphy

The unknown word guesser, *IceMorphy*, was designed to be a stand-alone module callable from different applications and to be used as an unknown word guesser in *IceTagger*. The purpose of *IceMorphy* is to generate the tag profile for a given word. This is fulfilled by either i) returning the tag profile for a known word found in a dictionary, or ii) guessing the tag profile for an unknown word. Moreover, *IceMorphy* is able to fill in the gaps of a tag profile (a gap signifies a missing tag in the profile) for words belonging to particular morphological (inflectional) classes.

IceMorphy uses a familiar approach for unknown word guessing, i.e. it performs morphological analysis, compound analysis and ending analysis. Since morphological analysis (and compound analysis) is more accurate than ending analysis, it is important to design a system where the main emphasis is on the former, and the latter is used when morphological analysis fails.

IceMorphy uses a general purpose dictionary, i.e. a dictionary derived from a tagged corpus. It is, therefore, not dependent on a base-form dictionary, as is the case, for example, with morphological analysers based on a two-level morphology.

5.2.1 The morphological analyser

The first component of the unknown word guesser is the morphological analyser. It tries to classify an unknown word as a member of a particular morphological class. A word belongs to a morphological class if the word's morphological ending is consistent with the inflection rules of the class. The current version uses 18 morphological classes for nouns, 5 classes for adjectives and 5 classes for verbs.

Basing the analysis on morphological classes is a common approach. It

is, for example, used in two German morphological systems which use a previously compiled stem dictionary for lookup (Lezius 2000, Nakov et al. 2003). However, the morphological analyser of *IceMorphy* is not dependent on a stem dictionary. Any general purpose dictionary, in which each word is tagged using the Icelandic tagset, will do, but the dictionary currently used is generated automatically from a tagged corpus.

For a given unknown word w, a morphological class is guessed based on the morphological ending of w. Then, the stem r of w is extracted and all k possible morphological endings for r are generated, resulting in search strings, s_i $(i = 1, \ldots, k)$, such that $s_i = r + ending_i$. A dictionary lookup is performed for s_i until a word is found having the same morphological class as was originally assumed or no match is found.

The morphological analyser tries, first, to classify the unknown word w as a verb, using the method described above. No matter if it succeeds or not, it next tries to classify w as a noun or an adjective, but not both. Therefore, w may be assigned both verb tags and noun/adjective tags.

A similar approach has been taken when automatically inducing rules for unknown word guessing in the work of Mikheev (1997), i.e. searching the dictionary for words that share the same stem, but have other morphological endings. However, the automatic rule induction method used to generate the rules for a tagger is very dependent on the training lexicon. Understandably, only rules that can be deduced by the lexicon will be produced. In contrast, the rules of *IceMorphy* are not dependent on a training lexicon since they are compiled using linguistic knowledge[1]. Another disadvantage with the automatic method is that it is not able to capture vowel mutation (e.g. "*kökur*" (cakes, plural) vs. "*kaka*" (a cake, singular)) which *IceMorphy* can handle because it is specifically tailored to Icelandic.

Consider the following example. Let us assume the word "*hesturinn*" (the horse; a masculine, singular, nominative case noun with a suffixed definite article) is an unknown word. Based on the suffix "*urinn*" the word is assumed to belong to the morphological class of regular masculine nouns. Consequently, the stem "*hest*" is extracted and all other inflectional endings for the stem are generated ("*hest-ur*", "*hest-*", "*hest-i*", "*hest-s*", "*hest-ar*", "*hest-a*", etc.). Hence, search strings, s_i, are generated, such that s_i="*hest*"+$ending_i$. A lookup is performed for each s_i, and if a lookup is successful for a given

[1]To some extent, the rules used by *IceMorphy* are dependent on a corpus, because a development corpus was used during development/testing of the rules.

search string, the corresponding tag(s) is returned.

Subsequently, modifications need to be performed on the tag returned. Let us assume a lookup is successful for the search string "*hesti*" (s_3) whose corresponding tag is "*nkeþ*". The fourth letter of a noun tag denotes the case and, since we assumed a nominative case for "*hesturinn*", we need to change the tag "*nkeþ*" to "*nken*". Furthermore, the fifth character for noun tags represents suffixed article and, therefore, the article letter ("*g*") needs to be added to this tag, resulting in the correct tag "*nkeng*".

If the above algorithm does not succeed in finding a morphologically related word using the ending of the original word as a guideline, then the following prefix handling is used. A prefix p_i (obtained from a hand-written list of common Icelandic prefixes; $(i = 1, \ldots, k)$) is prefixed to a given unknown word w. Thus, search strings, s_i, such that s_i=p_i+w $(i = 1, \ldots, k)$, are generated. A dictionary lookup is performed for each s_i until a word is found in the dictionary. If a word is found, then the original word w is assigned the same tag profile as was found for s_i.

The morphological analyser is, however, not flawless. To illustrate, consider what happens when analysing the word "*búar*" (neighbours). Based on the morphological ending "*r*" and the verb "*búa*" (to live), the analyser classifies this word as a third (or second) person singular verb (similarity exists, for example, "*ég borða*" (I eat), "*hann borðar*" (he eats); see table 3.8). Unfortunately, the third person singular form for this particular verb is "*býr*" because the verb is irregular.

5.2.2 The compound analyser

The second part of the unknown word guesser, the compound analyser, uses a straight-forward method of repeatedly removing prefixes from unknown words and performing a lookup for the remaining part of the word. If the remaining word part is not found in the dictionary it is sent to the morphological analyser for further processing. If the lookup or morphological analysis deduces a tag t for the remaining word part, the original word (without prefix removal) is given the same tag t.

To illustrate, consider the compound word "*nýfæddur*" (newborn). By removing the first two letters "*ný*" (a common Icelandic prefix (new)), a dictionary lookup is performed for the substring "*fæddur*". If *fæddur*" is found in the dictionary with tag t, the word "*nýfæddur*" is assigned the same tag t. Otherwise, "*fæddur*" is sent to the morphological analyser for further

processing.

As is the case for the the morphological analyser, the compound analyser can make mistakes. Consider, for example, the past participle "*upprisinn*" (risen up). The compound analyser will remove the prefix "*upp*" and perform a lookup for the remaining word "*risinn*", which, incidentally, is the masculine, singular, nominative noun "*risi*" (a giant) with a suffixed definite article. Thus, the analyser incorrectly classifies the word "*upprisinn*" as a noun.

5.2.3 The ending analyser

The third part of the unknown word guesser, the ending analyser, is called if an unknown word can be deduced neither by morphological analysis nor by compound analysis. This component uses a hand-written endings dictionary along with an automatically generated one. The former, which is consulted first, is mainly used to capture common endings for adjectives and verbs, for which numerous tags are possible. By only using an automatically generated list of endings from a tagged corpus, it is almost certain (unless the tagged corpus is enormous) that not all possible tags for given adjectives will be deduced.

The automatically generated ending dictionary is constructed in the following manner. From a tagged corpus, all possible word endings of length 1 to 5 are collected together with the corresponding tags (the minimum length of the remaining substring is 2 characters). We assume that the endings are different for capitalised words vs. other words and, therefore, produce two endings dictionaries, one for proper nouns and another for all other words. Endings that appear with less frequency than some specific threshold (10 in our case) are filtered out.

As pointed out earlier, ending analysis is less accurate than morphological/compound analysis. For example, for the word "*bleðillinn*" (the sheet), our ending analyser proposes the four tags "*nkeng_ nkeog_ lkensf_ lkeosf*", based on the "*llinn*" ending. However, only the first tag is correct for this particular word.

5.2.4 Default handling

If none of the above analysers is able to produce a tag profile for an unknown word, then a default mechanism is used.

Number tokens receive either the tag "*tp*" (percentage) or "*to*" (cardinal).

Other unknown words, which have not been assigned a tag by the previous modules, are considered i) common nouns (if not capitalised) and receive the tags "*nhen_ nheo_ nhfn_ nhfo*" (noun, neuter, singular/plural, nominative/accusative), or ii) proper nouns (if capitalised) and receive the tags "*nken-m_ nkeo-m_ nkeþ-m_ nkee-m*" (noun, masculine, singular, nominative/accusative/dative/genitive, person name).

5.2.5 Tag profile gaps

An important feature of *IceMorphy* is its handling of *tag profile gaps*. A tag profile gap arises when a particular word, listed in the dictionary, has some missing tags in its profile (set of possible tags). This, of course, presents problems to a disambiguator, since its purpose is to select one single correct tag from all possible ones. For each noun, adjective or verb of a particular morphological class, *IceMorphy* is able to fill in the gaps for the given word.

To illustrate this functionality, let us consider three examples. In all of them, we will use a dictionary D (stating word forms and corresponding tag profiles), automatically derived from the *IFD* corpus.

First, consider the noun "*afgreiðslu*" (handling). When this word form is looked up in D, only the tag "*nveo*" is found (denoting noun, feminine, singular, accusative). Based on the "*u*" morphological suffix (the stem is "*afgreiðsl*") and the accusative case of the tag, *IceMorphy* assumes the word belongs to a particular morphological feminine noun class, in which singular accusative, dative and genitive cases have the same word form. Consequently, *IceMorphy* generates the correct missing tags: "*nveþ*" and "*nvee*".

Second, consider the adjective "*leyndardómsfulla*" (mysterious). The lookup into D returns the tag profile "*lkeþvf_ lhenvf*" (these tags denote adjective, masculine, singular, dative, weak declension, positive; and adjective, neuter, singular, nominative, weak declension, positive, respectively). As stated in section 5.2.3, the ending analyser is used to capture common endings for adjectives. The word "*leyndardómsfulla*" is therefore sent to the ending analyser, which (based on the "*fulla*" ending) deduces eight other tags for the word. The resulting correct tag profile is: "*lkeþvf_ lhenvf_ lveosf_ lkfosf_ lkeovf_ lkeevf_ lvenvf_ lheovf_ lheþvf_ lheevf*"

Finally, consider the verb "*skrökva*" (lie), whose tag profile (found in D) is "*sfg3fn*". Based on the morphological suffix "*a*" (the stem is "*skrökv*") and this given tag, *IceMorphy* assumes that this word belongs to a particular

morphological verb class, in which the given word form is the same for i) indicative, active, third person, plural, present tense, ii) indicative, active, first person, singular, present tense, and iii) the infinitive. Thus, the two tags "*sfg1en_sng*", are added to the tag profile, resulting in "*sfg3fn_sfg1en_sng*".

The filling of tag profile gaps has a significant effect on the overall tagging performance. During evaluation, we found the overall average tagging accuracy increase by 0.85% when tag gap filling was used.

Note that, the above tag profile gap filling can of course be applied to a dictionary in a pre-processing step (off-line), instead of being run dynamically (on-line) during each lookup. On the other hand, the dynamic behaviour makes the system more flexible, since *IceMorphy* can be used with new dictionaries without having to worry about whether gap filling has been applied beforehand on the dictionary or not.

Moreover, this on-line vs. off-line choice for profile filling does not have any impact on the tags produced for a given dictionary D. The reason is the following. First, the exact same method (programming code) is used for on-line and off-line gap filling. Second, the profile filling mechanism for a word w in D, is only dependent on the current tag profile for w, but not on the profile for any other word. Hence, the fact that a word w' in D has had its profile already filled has no effect on the profile filling for w. Moreover, if w in D has already been "filled" off-line, then the on-line filling mechanism will simply try to add the exact same tags to the profile for w as were added off-line (adding duplicate tags has no effect).

5.3 IceTagger

IceTagger is a linguistic rule-based tagger (LRBT), designed for tagging Icelandic text[2].

In chapter 7, we will discuss previously published tagging results for Icelandic text — the tagging accuracy of the best performing DDT is only about 90.4%, when tested against the *IFD* corpus (Helgadóttir 2004). Apart from the desire to improve this relatively low accuracy[3], the following discussion motivated us to develop a LRBT for tagging Icelandic text.

[2] As apposed to a data-driven tagger (DDT) trainable on different languages.

[3] It is a good research methodology to continue working on a problem from where someone else left off.

5.3.1 Motivation

It has been argued that although trigram taggers have performed well for English the same might not necessarily be true for morphologically rich languages for which large tagged corpora are not available (Schmid 1995). The problem is data sparseness, i.e. the size of the tagset in relation to the size of the training data. 639 different tags occur in the *IFD* corpus and, even though the size of the training data (the corpus) is moderately large or about 590,000 tokens, the tagset size is very large in relation to the size of the training data. For example, 639 tags mean that $639^3 = 261$ million contextual parameters need to be estimated for a trigram tagger. Hence, on the average, only about 0.002 tokens are available per parameter!

A LRBT is not as sensitive to this data sparseness because its rules are not automatically derived from a tagged corpus, but, rather, hand-crafted using linguistic knowledge. Indeed, the rules are, to a certain extent, also dependent on a corpus — because they are compiled by examining phenomena extracted from it — but to a much lesser degree than a tagger based on a data-driven method.

Moreover, some of the errors made by a data-driven tagger are due to the limited window size used for disambiguation (e.g. three words in the case of a trigram tagger). Contrastingly, a LRBT can disambiguate focus words on the basis of words or tags which occur anywhere in the sentence, not only in the nearest neighbourhood.

Hence, one might expect that a carefully constructed LRBT should perform better than a data-driven tagger when tagging a morphologically complex language using a large tagset. The aim of developing *IceTagger* was, thus, to test the following research hypothesis: **Due to data sparseness, a higher tagging accuracy can be obtained by a linguistic rule-based tagger when tagging Icelandic text, than achieved by a state-of-the-art data-driven tagger. Moreover, this can be achieved without using an enormous effort in development of the tagging system**.

Disambiguation rules can be developed using the fact that a limited set of word forms is responsible for a large part of the total ambiguity. Using the *IFD* corpus, we found that the 30 most frequent ambiguous word forms account for 50% of the total ambiguity, and the 153 most frequent ambiguous word forms are responsible for 67% of the total ambiguity[4]. Interestingly, 21

[4]Total ambiguity $= \sum_{i=1}^{n} freq(w_i) * tags(w_i)$, where $freq(w_i)$ is the frequency of w_i in the given corpus and $tags(w_i)$ is the number of possible tags (i.e. the size of the tag

out of the 30 most frequent word forms (i.e. 70%) are pure function words or pronouns, i.e. these words do not belong to any other word classes than adverbs, conjunctions, the infinitive marker, prepositions or pronouns.

For French, it has been found that the 16 most frequent ambiguous word forms account for 50% of the total ambiguity and 97 most frequent ambiguous word forms are responsible for 67% of the total ambiguity (Chanod and Tapanainen 1995). Using this knowledge, one can concentrate on writing disambiguation rules for the most frequent ambiguous word forms.

Instead of relying only on disambiguation rules (which we call *local rules*; see Section 5.3.4), we assume that development time can be shortened and accuracy increased by using *heuristics* (see Section 5.3.5). The purpose of the heuristics is to tag grammatical functions and prepositional phrases, and use this information to force feature agreement where appropriate. The idea is that the use of the heuristics minimises the need to spend time on writing a large set of local rules.

5.3.2 Ambiguity and disambiguation

IceTagger consists of two phases: introduction of ambiguity (lexical phase) and disambiguation. In the former phase, the tag profile for each word, both known words (for which tags are sorted by descending frequency) and unknown words, is introduced. This is achieved with the help of a dictionary, automatically derived from the *IFD* corpus, and the unknown word guesser, *IceMorphy*.

Each word in a sentence to be tagged is looked up in a dictionary. If the word exists, i.e. the word is known, the corresponding tag profile for the word is returned. In the case of a tag profile gap, the unknown word guesser of *IceMorphy* is used for filling in the missing tags. If the word does not exist in the dictionary, i.e. the word is unknown, *IceMorphy* is used for guessing the tag profile. At the end of this phase, a given word of a sentence can have multiple tags, i.e. ambiguity has been introduced.

The purpose of the disambiguation phase is to eliminate as many inappropriate tags as possible from each word. The main characteristic of the disambiguation part of *IceTagger* is the use of only a small number of local rules (about 175) along with heuristics that perform further global disambiguation based on feature agreement. If, after local and global dis-

profile) for w_i. Only ambiguous words are taken into account.

ambiguation, a word is still not fully disambiguated, the most frequent tag, in the tag profile for the word, is selected. It can be argued that *IceTagger* is thus a combination of a rule-based tagger and a base tagger[5].

In the following sections, we describe the individual parts of the disambiguation phase. As will be described in section 5.5, the linguistic knowledge built into the disambiguation phase was developed using 10% of the *IFD* corpus. We call this subcorpus the *development corpus*.

5.3.3 Idioms and phrasal verbs

The first step of the disambiguation phase is to identify idioms, i.e. bigrams and trigrams which are always tagged unambiguously. Idioms are identified by examining lexical forms of adjacent words. The list of idioms was constructed semi-automatically in the following manner. First, we extracted automatically all trigrams in the *IFD* corpus that occurred at least ten times with the same tag sequence. Additionally, we hand-constructed a list of frequently occurring bigrams tagged unambiguously, by examining the development corpus.

The second step is to identify phrasal-verbs, whose words are adjacent in text. An Icelandic phrasal verb is a verb-particle pair (like "*fara út*" (go out)) where the particle is an adverb (because it is associated with a particular verb), but not a preposition. An automatically generated lexicon (from the *IFD* corpus) is used for recognising phrasal verbs.

5.3.4 Local rules

The third step of the disambiguation phase is the application of local elimination rules which perform disambiguation based on a local context (a window of 5 words; two words to the left and two words to the right of the focus word). The purpose of a local rule is to eliminate inappropriate tags from words. This reductionistic approach is common in rule-based taggers, and is, for example, used in CG systems.

In principle, the local rules are unordered. The firing of a rule is, however, dependent on the order of the words in a sentence. A sentence to be tagged is scanned from left to right and all tags of each ambiguous word are checked in a sequence. The tag profile for each (known) word is sorted in descending order

[5] A base tagger always selects the most frequent tag for each word.

of frequency and therefore the most frequent tag for each word is checked first. Depending on the word class (the first letter of the tag) of the focus word, the token is sent to the appropriate disambiguation routine which checks a variety of disambiguation constraints applicable to the particular word class and the surrounding words. At each step, only a single tag for the focus word is eliminated. This iterative process continues until the application of the rules do not result in a tag being eliminated in the given sentence.

The rules are written in a separate file. A Java-like syntax is used and the rules are compiled to Java code. The format of a local rule is:

RULE <condition>;

A *<condition>* is a boolean expression, whose individual components can refer to lexical forms or individual characters (word class/morphological features) of tags. If *<condition>* is true, then the tag in question for the focus word is eliminated. Let us now consider a couple of examples of local rules.

First, consider two rules which apply to preposition tags:

1. RULE nextToken.isOnlyVerbAny();
2. RULE currToken.lexeme.equalsIgnoreCase("um") AND nextToken.isNumeral();

In the first rule, the preposition tag is eliminated if the following token (nextToken) has only verb tags (isOnlyVerbAny()). To exemplify, consider the following sentence part: "*við vorum* " (we were). The word "*við*" can have the following five tags: "*ao_ aþ_ fp1fn_ aa_ nkeo*". These tags denote a preposition governing the accusative; a preposition governing the dative; a first person, plural, nominative personal pronoun; an adverb; and a masculine, singular, accusative noun, respectively. Since the following word is a verb, "*vorum*", this rule eliminates preposition tags in this context, leaving only the three tags "*fp1fn_ aa_ nkeo*".

The second rule refers to the lexical from of the current word. This rule applies, for example, to phrases like "*um 1930*". The word "*um*" has the tag profile "*ao_ aa*" (denoting a preposition governing the accusative and an adverb, respectively). The preposition tag does not apply when a numeral follows the word "*um*", and the above rule thus eliminates the preposition tag.

Secondly, consider two rules which apply to noun tags:

1. RULE prevToken.isOnlyWordClass(PREP) AND
   !tag.caseMatch(prevToken.getTags());

2. RULE prevToken.isOnlyWordClass(DEMPRONOUN) AND
   prevToken.isOnlyCase(NOMINATIVE) AND
   !tag.isCase(NOMINATIVE) ;

The first rule ensures that a case agreement holds between a preceding preposition tag (prevToken.isOnlyWordClass(PREP)) and the current noun tag. Thus, if the current tag does not agree in case with the preceding preposition (!tag.caseMatch(prevToken.getTags()), then this rule will eliminate the noun tag (assuming it is not the only remaining tag). To illustrate, consider the following preposition phrase: "*í borðið*" (in table). Let us assume that at a particular point in the disambiguation phase, the tag profile for the preposition "*í*" is "*ao*" (denoting a preposition governing the accusative) and the tag profile for the noun "*borðið*" is "*nheog_nheng*" (denoting neuter, singular, accusative/nominative, with suffixed definite article). Then the above rule will eliminate the second tag of the noun.

The second rule eliminates oblique case tags (!tag.isCase(NOMINATIVE)) of a noun when the preceding word is a demonstrative nominative pronoun (prevToken.isOnlyWordClass(DEMPRONOUN) AND . . .). For example, for the phrase "*sú vist*" (that stay), this rule eliminates the last two of the three possible tags, "*nven_nveo_nveþ*" (nominative, accusative and dative) for the noun "*vist*" because the demonstrative pronoun "*sú*" is in the nominative case (thus enforcing case agreement).

A number of functions exists for the rule writer to use on tokens and tags to build <condition>. Some of the most useful functions are:

- *token.isWordClass(aWordClass)*: Returns true if *token* has a tag denoting the word class *aWordClass*.

- *token.isOnlyWordClass(aWordClass)*: Returns true if *token* has only tags denoting the word class *aWordClass*.

- *token.isCase(aCase)*: Returns true if *token* has a tag with the case feature *aCase*.

- *token.isOnlyCase(aCase)*: Returns true if *token* has only tags with the case feature *aCase*.

- *tag.isCase(aCase)*: Returns true if *tag* has the case feature *aCase*.
- *token.caseMatch(aToken)*: Returns true if *token* agrees in case with any tag of the token *aToken*
- *tag.caseMatch(aTagProfile)*: Returns true if *tag* agrees in case with any tag in the tag profile *aTagProfile*
- *token.genderNumberCaseMatch(aToken)*: Return true if *token* agrees in gender, number and case with any tag of the token *aToken*.
- *token.personNumberMatch(aToken)*: Return true if *token* agrees in person and number with any tag of the token *aToken*.
- *token.isOnlyVerbAny()*: Returns true if *token* has only tags denoting verbs.
- *token.isVerbActive()*: Returns true if *token* has a tag denoting a verb and the active feature.
- *token.isVerbSubjunctive()*: Returns true if *token* has a tag denoting a verb and the subjunctive feature.
- *token.isVerbInfinitive()*: Returns true if *token* has a tag denoting a verb and the infinitive feature.
- *token.isVerbPastParticiple()*: Returns true if *token* has a tag denoting a verb and the past participle feature.
- *token.isVerbSupine()*: Returns true if *token* has a tag denoting a verb and the supine feature.
- *token.isAdjectivePositive()*: Returns true if *token* has a tag denoting an adjective and the positive feature.
- *token.isAdjectiveComparative()*: Returns true if *token* has a tag denoting an adjective and the comparative feature.
- *token.isAdjectiveSuperlative()*: Returns true if *token* has a tag denoting an adjective and the superlative feature.
- *token.isPunctuation()*: Returns true if *token* has a tag denoting a punctuation.

- *token.lexeme.equalsIgnoreCase(aString)*: Returns true if the lexeme of the tokens equals (ignoring case) the string *aString*.

- *token.lexeme.endsWith(aString)*: Returns true if the lexeme of the tokens ends with the string *aString*.

5.3.5 Heuristics

Once local disambiguation has been carried out, each sentence is sent to a global heuristic module. The heuristics are a collection of algorithmic procedures whose purpose is to tag grammatical functions and prepositional phrases (PPs), and to force feature agreement where appropriate. We call these heuristics global because, when disambiguating a particular word, a heuristic can refer to another word which is not necessarily in the immediate neighbourhood. Similar heuristics to those described below, may be applied to other morphologically complex languages.

Before the heuristics are applied, each sentence is partitioned into clauses using tokens like comma, semicolon and coordinating/relative conjunctions as separators (care is taken not to break up enumerations into individual parts). The heuristics then repeatedly scan each clause and perform the following in order:

1. mark PPs
2. mark verbs
3. mark subjects
4. force subject-verb agreement
5. mark objects
6. force subject-object agreement
7. force verb-object agreement
8. force nominal agreement
9. force PP agreement

We will now consider each heuristic above in turn as well as briefly describe other miscellaneous heuristics. Recall that, before the heuristics are run, local rules have been applied and the tag profile for each known word is sorted by descending frequency.

We will use the following main illustrative sentence: "*gamli maðurinn borðar kalda súpu með mjög góðri lyst*" (old man eats cold soup with very good appetite)[6]. After introduction of ambiguity and the application of local disambiguation rules by *IceTagger*, the words of this sentence have the following tags:

(1) *gamli*/**lkenvf** *maðurinn*/**nkeng** *borðar*/**sfg3en_sfg2en**
kalda/**lhenvf_lkfosf_lveosf_**
lkeþvf_lheþvf_lheovf_lheevf
súpu/**nveo_nveþ_nvee** *með*/**aþ_aa**
mjög/**aa** *góðri*/**lveþsf** *lyst*/**nveþ_nveo_nven**[7]

5.3.5.1 Marking prepositional phrases

The first heuristic searches for words, in the current clause, that have a preposition tag as their first (i.e. most frequent) remaining tag. Each such word is assumed to be a preposition and therefore all non-prepositional POS tags for the word are removed. Additionally, the word is marked with a *PP* tag. Nominals following the assumed preposition are marked with a *PP* tag as well, if there is a case feature agreement match between the nominals and the preposition.

In (1), each word (with the exception of the adverb) in the PP "*með mjög góðri lyst*" is marked with a *PP* tag, resulting in the following POS and syntactic tags:

(2) *með*/**aþ** *PP* *mjög*/**aa** *góðri*/**lveþsf** *PP*
lyst/**nveþ_nveo_nven** *PP*

5.3.5.2 Marking verbs

When marking verbs in the current clause, words are searched that have a verb tag as their first remaining tag. Each such word is assumed to be a verb

[6] When translating examples to English we use word by word translation.

[7] **sfg3en**/**sfg2en**: verb, indicative, active, $3^{rd}/2^{nd}$ person, singular, present tense., **aþ**: preposition governing dative, **aa**: adverb. See table 4.5 for the semantics of the noun and the adjective tags.

and therefore all non-verb POS tags for the word are removed. Each verb found is marked with a functional verb tag *VERB*.

In (1), "*borðar*" is marked with the tag *VERB*.

5.3.5.3 Marking subjects of verbs

The third heuristic marks the single closest subject of a given verb, i.e. in most cases the head (a noun) of a subject noun phrase (NP). Since Icelandic word order is relatively free, both "*Jón gaf eina bók*" (*John gave one book*) and "*eina bók gaf Jón*" (*one book gave John*) are possible. The heuristic thus assumes that subjects can be found either preceding or following the verb.

For each verb v, already marked with a *VERB* tag, the tokens are first scanned starting from the left of v (since SVO order is more likely than OVS order). If the immediate token to the left of v is a relative conjunction or a comma, then it is assumed that the subject can be found in the previous clause (see below). Otherwise, if the current token is a nominal (not marked with a *PP* tag) and it agrees with v in person and number, it is marked with a functional tag *SUBJ* — if not, the scanning continues (to the left) until the beginning of the clause is reached.

If no subject candidate is found to the left of v, a search continues using the next two tokens to the right of v (it is thus assumed that subjects appearing further away to the right are unlikely), using the same feature agreement criterion as before.

If at this point a subject candidate has still not been found, a search is performed in the previous clause, and the first nominal found is then marked with a *SUBJ* tag (if it is not already marked as an object of a verb in an earlier clause).

In (1), "*maðurinn*" is marked as a subject because it agrees with the verb "*borðar*" in person and number (notice that the modifier "*gamli*" is not marked — the heuristic described in section 5.3.5.8 will force an agreement between modifiers and heads of NPs), resulting in the following:

(3) *gamli/***lkenvf** *maðurinn/***nkeng** *SUBJ borðar/***sfg3en_sfg2en** *VERB*

5.3.5.4 Forcing subject-verb agreement

Once verbs and subjects of verbs have been identified, feature agreement is forced between the respective words.

In (3), this means removing the second person tag from the verb "*borðar*" because the subject "*maðurinn*" is third person. Moreover, if the subject is in the nominative case (which is generally the case except for subjects of special verbs that demand oblique case subjects) all non-nominative cases are removed from the subject.

5.3.5.5 Marking objects of verbs

This heuristic marks direct objects and verb complements. Both types receive the same functional tag *OBJ*. For each verb already marked with a *VERB* tag, a search is performed for objects following the verb or, if the search is unsuccessful, for objects preceding the verb.

Objects can be nominals (direct objects or complements) or past participle verbs (only complements). When searching for nominals, words which have already been marked with *PP* or *SUBJ* tags are ignored. Only the last word in a sequence of nominals is marked. Effectively, in most cases, this means that only the head of an NP is marked as an object. For the purpose of enforcing feature agreement between adjacent nominals, marking the head is sufficient because, as previously stated, internal NP agreement is forced by the heuristic described in section 5.3.5.8.

In (1), the noun of the NP "*kalda súpu*" is marked as an object and the whole sentence now has the following tags:

(4) *gamli/***lkenvf** *maðurinn/***nkeng** *SUBJ borðar/***sfg3en** *VERB*
*kalda/***lhenvf_lkfosf_lveosf_**
lkeþvf_lheþvf_lheovf_lheevf
*súpu/***nveo_nveþ_nvee** *OBJ með/***aþ_aa** *PP*
*mjög/***aa** *góðri/***lveþsf** *PP lyst/***nveþ_nveo_nven** *PP*

5.3.5.6 Forcing subject-object agreement

In Icelandic, feature agreement is needed between a subject and a verb complement. For example, in sentences like "*Jón er fallegur*" and "*María er falleg*" (*John/Mary is beautiful*), the complement adjusts itself to the subject. Additionally, a subject-object agreement is needed where the object is a reflexive pronoun, e.g. "*Jón meiddi sig*" (*John hurt himself*).

This heuristic forces both gender and number agreement between a subject and a complement, and between a subject and an object in the form of a reflexive pronoun.

5.3.5.7 Forcing verb-object agreement

Icelandic verbs govern the case of their direct objects which is, generally, either accusative or dative. However, a verb complement is always in the nominative case. The correct case of a direct object must be "learnt" for each verb because no general rule applies. For example, "*Jón gaf bókina*" (accusative object; *John gave book*) is correct but not "*Jón henti bókina*", but rather "*Jón henti bókinni*" (dative object; *John threw book*).

A lookup table, automatically derived from the *IFD* corpus, is used for determining the correct case for direct objects (this table, thus, provides partial verb subcategorisation information). A lookup is performed for a given verb lexeme and the correct case is returned. Tags of the associated object that do not include the correct case are then removed. If the lookup is unsuccessful, and the marked object is not a complement, then only the nominative case tags are removed from the object[8].

In (4), the verb "*borðar*" demands an accusative object (found with a lookup) and, as a result, all non-accusative case tags are removed from the object "*súpu*". After this removal, the sentence part "*borðar kalda súpu*", thus, contains the following tags:

(5) *borðar/***sfg3en** *VERB*
*kalda/***lhenvf_lkfosf_lveosf_**
lkeþvf_lheþvf_lheovf_lheevf
*súpu/***nveo** *OBJ*

5.3.5.8 Forcing agreement between nominals

Agreement in gender, number and case between a noun and its modifiers is a characteristic of Icelandic NPs. This heuristic forces such an agreement in the following manner. Starting at the end of a clause, it searches for a nominal n, i.e. a head of a NP. If a head is found, the heuristic searches for modifiers to the left of n (care must be taken not to step inside a PP phrase if n itself is not part of that PP phrase). Agreement is forced between the head and its modifiers by removing inappropriate tags from either word.

In (5), the heuristic removes the six tags **lhenvf_lkfosf_lkeþvf_lheþvf_lheovf_lheevf** from the adjective "*kalda*", in order to force gender, num-

[8] In this case, as discussed in Section 5.3.5.10, the most frequent tag of the objects is used.

ber and case agreement with the tags of the following noun "*súpu*" (feminine, singular, accusative). Additionally, this heuristic removes the tags **nveo_nven** from the noun "*lyst*" (see (2)) because of the feature agreement with the preceding adjective "*góðri*". Notice that an agreement already holds in the first NP, "*gamli maðurinn*" (see (3)). After these tag eliminations, the final disambiguated sentence looks like:

(6) *gamli/***lkenvf** *maðurinn/***nkeng** *SUBJ borðar/***sfg3en** *VERB*
*kalda/***lveosf** *súpu/***nveo** *OBJ*
*með/***aþ** *PP mjög/***aa** *góðri/***lveþsf** *PP lyst/***nveþ** *PP*

There are some special cases regarding adjacent nominals that need to be accounted for. For example, when two nouns are adjacent to each other, one of them is, generally, in the genitive case, e.g. "*fjölskylda mannsins(genitive)*" (*family man's*) or "*vesalings(genitive) konan*" (*poor woman*). This heuristic makes sure to remove non-genitive tags from a noun where appropriate.

5.3.5.9 Forcing prepositional phrase agreement

The last main heuristic forces feature agreement in preposition phrases. Two things need to be accounted for. First, when a preposition has two possible case tags, i.e. accusative and dative tags (which is common for prepositions like "*á, eftir, fyrir, í, með*" (*on, after, for, in, with*)), the heuristic removes one of the case tags based on the case of a following word in the PP.

If the following word does not unambiguously select the correct tag for the preposition then a search is performed for a preceding verb. A verb-preposition pair usually unambiguously determines the correct case of the preposition[9]. For example, in the sentence "*Jón settist á plötu*" (*John sat-down on board*) the verb-preposition pair "*settist á*" determines an accusative case for the preposition "*á*". In contrast, in the sentence "*Jón lá á plötu*" (*John lay on board*) the pair "*lá á*" determines a dative case for the preposition "*á*". In this case, a lookup table, automatically derived from the *IFD* corpus, is used for determining the correct case of the preposition. A lookup is performed for a given verb-preposition lexeme, the correct case returned and the conflicting tag of the preposition is removed. If the lookup is unsuccessful the most frequent tag of the preposition is used.

[9] A verb which is followed by a preposition (phrase) is called a prepositional verb.

Secondly, once the correct preposition case tag is determined, a case agreement between the preposition and the rest of the words in the PP is forced. This is straight-forward, since the correct case is now known and the words to search for have already been marked by the heuristic described in section 5.3.5.1.

This heuristic does not have any effect on our example sentence because the sentence is, at this point, already fully disambiguated.

5.3.5.10 Other miscellaneous heuristics

In addition to the above main heuristics, specific heuristics are used to choose between supine and past participle verb forms, infinitive or active verb forms, and ensuring agreement between reflexive pronouns and their antecedents.

Finally, for words that still have not been fully disambiguated, the default heuristic is simply to choose the most frequent tag. Consequently, *IceTagger* always performs full disambiguation.

5.3.6 Special verbs

Some Icelandic verbs have special characteristics. *IceTagger* keeps auxiliaries, the verbs "*vera/verða*" (be/become), and verbs that demand non-nominative case subjects in a special (base) dictionary (all possible word forms for these verbs are included in the dictionary). These verbs are marked in a special way to facilitate correct disambiguation decisions when encountered. The special marking, for example, includes information on which case a verb demands for its (non-nominative case) subject. As another example, tags for word forms for the verbs "*vera/verða*" are marked with a special code, which tells the tagger to expect a predicative nominative for this verb. This mechanism simplifies rule writing, because rules do not need to be written for each word form of the given verb. Instead the rules can refer to the special codes included in the corresponding tags.

5.4 TriTagger

We have re-implemented (in Java and Perl) the functionality of the *TnT* tagger (see section 2.3.4.1), and hereafter refer to this tagger as *TriTagger*. The difference between these two taggers is that *TriTagger* uses the same

list of idioms as *IceTagger*, and the special dictionary described above as a backup dictionary. We use this tagger for integration with *IceTagger* (see section 7.3).

TriTagger is a statistical tagger based on an HMM. The tagger is data-driven, i.e. it learns its language model from a pre-tagged corpus. *TriTagger* uses the tokenisation method described in section 5.1. The algorithm used by the tagger is as follows (consult (Brants 2000) for full details):

Known words are handled in the manner described in section 5.3. Since *TriTagger* is language independent, it has no knowledge of Icelandic morphology. Suffix analysis is, therefore, the default method for guessing possible tags for unknown words.

On the other hand, since *IceMorphy* already exists, it can be called from within *TriTagger*. In that case, if *IceMorphy* can use morphological/compound analysis (as opposed to ending analysis or default handling) to guess tags for an unknown word w, then *TriTagger* will use those tags, with lexical probabilities, $p(w|t_j)$, $j = 1 \ldots n$, set to $1/n$. For other unknown words, suffix analysis is carried out.

The probabilities of the model are estimated from a training corpus using maximum likelihood estimation. Therefore, *TriTagger* needs to be trained on a pre-tagged corpus before it can be used.

5.5 The development phase

In the beginning, we decided to implement the system using the Java programming language. There were mainly two reasons for this decision.

First, the main advantage of the Java language is the portability of applications, i.e. since the language is interpreted by the Java Virtual Machine (JVM) at run-time, re-compilation of source code is not necessary when the applications are executed on different platforms.

Secondly, in recent years, Java has been used to a greater extent in the NLP community. For example, the widely used *GATE* (General Architecture for Text Engineering) system (Cunningham 2000) is implemented in Java. In our view, this increased usage of Java in NLP will simplify the integration of *IceNLP* with other NLP systems.

Logically, we started by implementing the tokeniser, in which various scenarios needed to be accounted for, as described in section 2.2.

In the next step, *IceMorphy* and *IceTagger* were implemented in parallel by an iterative process. This process consisted of writing local rules and heuristics (for *IceTagger*) and procedures for morphological and ending analysis (for *IceMorphy*), with the purpose to minimise the error rate when tested against the tenth test corpus (our development corpus) of the *IFD* corpus. At each step the most serious errors were corrected, by additions and/or modifications to the source code, the errors examined, and the process started over again. The local rules are not part of the source code itself, but are instead kept in a separate file, modifiable by a user. A Java-like code is used to write rules and a rule compiler (developed by us) is then used to transform the rule code to Java code.

Even though the tenth corpora was used for development and testing, care was taken to implement general linguistic rules that would be applicable to any type of Icelandic text. Nevertheless, it is very likely that such a process will result in a lower tagging accuracy when tested against text not used for development, as we will demonstrate in section 7.2.5.

In the third step, we implemented *TriTagger* in Java. As discussed in section 5.4, the tagger is a re-implementation of the *TnT* tagger.

Thus, the whole tagging system is written as a collection of Java classes — totalling about 17,000 lines of source code.

5.6 Discussion

As discussed in section 2.3.5, linguistic rule-based methods are developed with the purpose of tagging a specific language using a particular tagset. This fact makes it relatively easy to tailor the method to the language at hand, which, in contrast, can be difficult to do when using data-driven language independent methods. This advantage does indeed hold for our linguistic rule-based system, specifically *IceMorphy* and *IceTagger*.

On the other hand, the advantage of our system is also its main disadvantage. Our system was developed for tagging a specific language, Icelandic, and the emphasis was not to make it usable for tagging other languages. Our system is thus not a language independent formalism like, for example, the CG framework.

A natural question to ask is then: Why did we not use CG for building our linguistic rule-based system? There are mainly two reasons. First, experience has shown that the construction of a CG for a particular language

is a demanding and a time-consuming task. This is mainly due to the effort needed to construct a morphological analyser/lexicon using the framework of two-level morphology, and, additionally, due to the number of constraints needed for acceptable performance. This would probably not be suitable for a PhD project, and, moreover, the degree of originality would be questionable. We wanted to try to improve the tagging accuracy of Icelandic text by developing a LRBT without using an enormous time for development. Secondly, we wanted to investigate the effectiveness of mixing local rules with heuristics (global rules), which is not possible in CG.

Despite the fact that our tagging system was designed to tag Icelandic text using a particular tagset, some effort has been made to support language and tagset independence. The local rules are written (using Java-like syntax) in a separate file, which is compiled into Java code. The rules are thus separated from the program code. In section 5.3.4, we described the format of the rules and listed some of the functions available for use by the rule writer. The rule language is a kind of an application programming interface (API) for the rule writer, and the underlying natural language being processed is completely "hidden". If, for example, the underlying language does not have a case feature then the rule writer would simply not use the functions available that deal with the case feature, like *token.isCase(aCase)*, *token.caseMatch(aToken)*, etc.

Our system currently uses the main (large) Icelandic tagset. Functions that query tags or individual features of tags are encapsulated in a separate Java class, the *Tag* class. Making the system work for another tagset (for Icelandic or any other language) is thus a matter of changing the *Tag* class (and possibly adding new member functions to it) and recompile the system. It can thus be argued that our local rule formalism is partly language independent. Of course, a new morphological analyser (unknown word guesser) would be needed in the system if it were going to be used for another language.

In contrast, the heuristics used by our system are not language independent. They are a collection of algorithmic procedures whose purpose is to tag grammatical functions and prepositional phrases, and to force feature agreement where appropriate. The procedures are very much dependent on the word order in Icelandic and the specific feature agreement rules inside phrases and between words. Tailoring the heuristics to work for another language would thus be a significant task.

It is important to note the difference between our heuristics and what is

called "heuristic disambiguation constraints" in CG. First, the latter is used as "riskier constraints", but are formulated just like ordinary constraints (see (Karlsson et al. 1995) for a thorough description), whereas our heuristics are totally different from ordinary (local) constraints. The heuristic disambiguation constraints are, however, as in our system, applied only when the ordinary (local) constraints find no more applications. Secondly, heuristic disambiguation constraints in CG work just on the small share of ambiguity left after "safe" disambiguation, whereas our heuristics actually work on a large share of the ambiguity left after the application of (relatively few) local rules. The default heuristic in our system, which simply chooses the most frequent tag from the remaining ones for each token, is a kind of a "riskier constraint", which is applied on the small amount of ambiguity left after application of the local rules and the general heuristics[10].

[10]Such probabilistic heuristics can be implemented as an add-on in CG.

Chapter 6

The Parsing System

Our parsing system consists of a shallow annotation scheme and a shallow parser, based on finite-state techniques, called *IceParser*, for parsing Icelandic text[1]. The parser accepts POS-tagged text as input and returns as output the same text syntactically annotated with brackets and labels, denoting constituent structure and grammatical functions.

This chapter is organised as follows. In Section 6.1, we describe the motivation for our work on the shallow parser. Section 6.2 describes our parsing scheme and, in Section 6.3, we describe the implementation of the parser.

6.1 Motivation

Our motivation for developing a finite-state parser for the Icelandic language is the following. First, as we discussed in Section 2.4.7, there is a scarcity of available parsers for the Icelandic language. One full parser has been developed that is not publicly available. This lack of parsing tools has become a hindrance to further research and development work on NLP for the Icelandic language. The development of *IceParser* is a step towards the goal of developing and making basic NLP units available for the research community.

Secondly, we have previously mentioned (see Section 2.4.2) that in many NLP applications it can be sufficient to analyse sentence parts or phrases instead of performing a full parse. Partial parsing is, for example, sufficient

[1]This research was funded in 2006 by the Icelandic Research Fund under contract "Shallow parsing of Icelandic text".

in systems for information extraction, some form of grammar correction, question-answering and shallow corpus annotation. Moreover, finite-state parsers are robust and efficient.

Thirdly, no treebank exists for Icelandic, and, thus, using data-driven parsing methods is currently not an option.

Fourth, despite the fact that Icelandic has a relatively free word order, its rich case system should help during the parsing process. In particular, the case feature should help when writing patterns for grouping words into phrases and identifying syntactic functions.

Lastly, we hypothesise that **the use of a finite-state parsing method for a morphologically complex language, with a relatively free word order, like Icelandic, is effective, and additionally, that an enormous effort is not needed in the development of a finite-state parser for the language in order to obtain good results.**

6.2 A shallow parsing scheme

In this section, we propose a shallow annotation scheme for Icelandic text[2]. By shallow annotation, we mean that syntactic structures are rather flat and simple, i.e. the main emphasis is to annotate core phrases without showing a complete parse tree.

With reference to the EAGLES guidelines (see Section 2.4.4.1), our scheme consists of brackets and labels indicating constituent structure and functional relations (syntactic functions). Our scheme, thus, follows the dominant paradigm in treebank annotation, i.e. it "[is a] kind of theory-neutral annotation of constituent structure with added functional tags" (Nivre 2002).

We assume that the text to be annotated has already been POS tagged using the tagset described in Section 4.2. This tagset includes both word class and morphological information.

At the end of the section, we describe a *grammar definition corpus* (GDC; see Section 2.4.4) annotated using our scheme.

[2]This annotation scheme was designed in collaboration with Eiríkur Rögnvaldsson, Professor of Icelandic Language, at the University of Iceland.

6.2.1 Constituent structure

The EAGLES guidelines recommend annotation of the following constituent categories: sentence, clause, noun phrase, verb phrase, adjective phrase, adverb phrase and prepositional phrase. Since our annotation scheme puts emphasis on core phrases, we neither include sentence nor clause categories.

We use brackets and labels to indicate constituents. Two labels are attached to each marked 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 syntactic annotation (denoting adverb, adjective, noun, prepositional and verb phrase, respectively). Additionally, we use the labels **CP**, **SCP**, **InjP**, **APs**, **NPs** and **MWE** for marking coordinating conjunctions, subordinating conjunctions, interjections, a sequence of adjective phrases, a sequence of noun phrases, and multiword expressions, respectively. Hence, in our scheme, every word is a part of some constituent structure.

In the following sections, we describe the structure of each constituent in more detail. For each constituent, we show examples obtained from our GDC. The English gloss (most often a word-by-word translation) appears in parenthesis with most of the examples. For saving space, we leave out the POS tag associated with each word in the examples.

6.2.1.1 Multiword expression phrases

A multiword expression (MWE) phrase comprises fixed multiword expressions which function as a single word. We 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).

Below we show 2-3 examples of each kind:

1. [MWE_CP eins og MWE_CP] (as)
2. [MWE_CP til að MWE_CP] (in order to)
3. [MWE_CP á meðan MWE_CP] (while)
4. [MWE_AdvP hvers vegna MWE_AdvP] (why)
5. [MWE_AdvP allt í einu MWE_AdvP] (suddenly)

6. [MWE_AdvP til dæmis MWE_AdvP] (for example)
7. [MWE_AP alls konar MWE_AP] (all kinds of)
8. [MWE_AP hvers kyns MWE_AP] (every kind of)
9. [MWE_PP fyrir framan MWE_PP] (in front of)
10. [MWE_PP út í MWE_PP] (out into)
11. [MWE_PP innan um MWE_PP] (among)

In example no. 9, the preposition ("*fyrir*") precedes the adverb ("*framan*"), but in examples no. 10 and 11 the adverbs ("*út*", "*innan*") precede the prepositions ("*í*", "*um*").

We have compiled a list of multiword expressions for each of the different kinds of MWEs (see Appendix C.4).

6.2.1.2 Adverb phrases

An adverb phrase ([AdvP ... AdvP]) consists of a sequence of one or more adverbs. The following are examples of adverb phrases:

1. [AdvP ekki AdvP] (not)
2. [AdvP svo AdvP] (so)
3. [AdvP þar AdvP] (there)
4. [AdvP þó AdvP] (although)
5. [AdvP þar með AdvP] (thereupon)
6. [AdvP í gær AdvP] (yesterday)
7. [AdvP þá fyrst AdvP] (then first)
8. [AdvP ekki síst AdvP] (not least)

Note that two (or more) adjacent adverbs are not necessarily part of the same adverb phrase. For example, consider the sentence *"skólar byrja bráðum aftur"* (schools start soon again). The correct annotation includes the two separate adverb phrases *[AdvP bráðum AdvP]* and *[AdvP aftur AdvP]*, but

not the single adverb phrase *[AdvP bráðum aftur AdvP]*. The reason is that the former adverb can be moved around in the sentence (without having to move the other adverb), e.g. resulting in a sentence like *"bráðum byrja skólar aftur"*.

6.2.1.3 Conjunction phrases

We distinguish between two types of conjunctions: coordinating conjunctions [CP ... CP]) and subordinating conjunctions [SCP ... SCP]). Only the following seven conjunctions are classified as coordinating conjunctions: *"og"* (and), *"en"* (but), *"eða"* (or), *"enda"* (because), *"heldur"* (but), *"ellegar"* (or else), *"né"* (nor).

A conjunction phrase consists of one conjunction. The following are examples of conjunction phrases:

1. [CP og CP] (and)
2. [CP en CP] (but)
3. [SCP sem SCP] (that/who/which)
4. [SCP að SCP] (that)
5. [SCP þegar SCP] (when)

6.2.1.4 Interjection phrases

An interjection phrase ([InjP ... InjP]) consists of one interjection. The following are examples of interjection phrases:

1. [InjP hí InjP] (heh)
2. [InjP æ InjP] (ouch)
3. [InjP takk InjP] (thanks)
4. [InjP já InjP] (yes)

6.2.1.5 Adjective phrases

An adjective phrase ([AP ... AP]) consists of an adjective, optionally preceded by a modifying adverb phrase. The following are examples of adjective phrases:

1. [AP erfitt AP] (difficult)
2. [AP kalt AP] (cold)
3. [AP meira AP] (more)
4. [AP [AdvP mjög AdvP] erfitt AP] (very difficult)
5. [AP [AdvP ákaflega AdvP] fágætur AP] (extremely rare)

The first three examples show adjective phrases consisting of a single adjective, while examples no. 4-5 demonstrate adverb phrases embedded in adjective phrases.

6.2.1.6 A sequence of adjective phrases

A sequence of adjective phrases ([APs ... APs]) consists of two or more consecutive adjective phrases (optionally separated by a CP or a comma) agreeing in gender, number and case. A sequence of such phrases typically denotes an enumeration of some kind. The following are examples of such sequences:

1. [APs [AP lágreist AP] [AP svört AP] APs] (low-rise black)
2. [APs [AP þrekinn AP] [CP og CP] [AP mikill AP] APs] (beefy and large)
3. [APs [AP stórar AP] [CP eða CP] [AP litlar AP] APs] (big or small)
4. [APs [AP gula AP] , [AP veðraða AP] APs] (yellow, weatherworn)
5. [APs [AP vörpulegur AP] , [AP skarpleitur AP] [CP og CP] [AP svipsterkur AP] APs] (pretty, sharp-featured and strong-looked)
6. [APs [AP [AdvP jafnan AdvP] grá AP] [CP eða CP] [AP skjöldótt AP] APs] (usually gray or multi-coloured)

6.2.1.7 Noun phrases

The structure of a noun phrase ([NP ...NP]) is the most complicated of all the phrases. In general, the unmarked word order in a noun phrase headed by a noun is: indefinite pronoun, demonstrative pronoun/article, numeral, adjective phrase, noun (and a possessive pronoun). This word order is relatively fixed with some exceptions (see below). Noun phrases can also consist of a single (personal, demonstrative, indefinite, or interrogative) pronoun.

Number, gender and case agreement holds between the words of a noun phrase.

The list below shows some examples of noun phrases:

1. [NP ég NP] (I)
2. [NP sig NP] (himself/herself)
3. [NP allt NP] (all)
4. [NP þetta NP] (this)
5. [NP hvað NP] (what)
6. [NP maður NP] (man)
7. [NP 1954 NP]
8. [NP Stefán NP]
9. [NP Einar Þorgilsson NP]
10. [NP sjálfan mig NP] (myself)
11. [NP þrír fingur NP] (three fingers)
12. [NP árið 1982 NP] (year 1982)
13. [NP pabbi þinn NP] (father your)
14. [NP þetta kvöld NP] (this evening)
15. [NP [AP gömul AP] húsgögn NP] (old furniture)
16. [NP [AP nýkjörinn AP] forseti NP] (newly-elected president)

17. [NP [AP [AdvP liðlega AdvP] þrítugur AP] karlmaður NP] (a-little-more-than thirty man)

18. [NP allt þetta [AP þunga AP] vatn NP] (all this heavy water)

19. [NP enginn [AP venjulegur AP] maður NP] (no ordinary man)

20. [NP hinn [AP gagnrýni AP] efnafræðingur NP] (the critical chemist)

21. [NP þessu [AP fyrsta AP] tölublaði NP] (this first issue)

22. [NP þessi [APs [AP brúnu AP] , [AP saklausu AP] APs] augu NP] (these brown, innocent eyes)

23. [NP [APs [AP gula AP] , [AP veðraða AP] APs] múrveggnum NP] (yellow, weatherworn brick-wall)

24. [NP [APs [AP ungi AP] [CP og CP] [AP glæsilegi AP] APs] organistinn NP] (young and elegant organist)

25. [NP þess [APs [AP þriðja AP] [AP stærsta AP] APs] NP] (the third biggest)

Examples no. 1-8 show noun phrases consisting of a single word. The first two include a personal pronoun, the third an indefinite pronoun, the fourth a demonstrative pronoun, the fifth an interrogative pronoun, the sixth a common noun, the seventh a numeral and the eighth a proper noun.

Examples no. 9-14 demonstrate noun phrases comprising two words and examples no. 15-25 show adjective phrases included in noun phrases.

Some exceptions to the main word order need to be accounted for. Below we present two examples of these exceptions:

1. [NP maður einn NP] (man one)

2. [NP sinn [AP sterkasta AP] bakhjarl NP] (his strongest sponsor)

In the first example, the indefinite pronoun follows the noun (instead of preceding it), and in the second sentence the possessive pronoun precedes the adjective/noun (instead of following it).

6.2.1.8 A sequence of noun phrases

A sequence of noun phrases ([NPs ... NPs]) consists of two or more consecutive noun phrases (optionally separated by a CP and/or a comma) agreeing in case. Moreover, a sequence of noun phrases can include a qualifier noun phrase which follows (or precedes) another noun phrase. A sequence of noun phrases, typically, denote an enumeration of some kind. The following are examples of noun phrase sequences:

1. [NPs [NP þrumur NP] [CP og CP] [NP eldingar NP] NPs] (thunder and lightning)
2. [NPs [NP þeim hugleiðingum NP] [CP og CP] [NP því starfi NP] NPs] (those speculations and that job)
3. [NPs [NP [AP gömul AP] húsgögn NP] , [NP [AP latneskar AP] bækur NP] [CP og CP] [NP smyrðlinga NP] NPs]
4. [NPs [NP fiskum NP] , [NP liðdýrum NP] [CP og CP] [NP spendýrum NP] NPs] (fish, arthropods and mammals)
5. [NPs [NP börn NP] [NP hans NP] [CP og CP] [NP niðjar NP] NPs] (children his and descendants)
6. [PP við [NPs [NP Lyme NP] [NP flóa NP] NPs] PP] (at Lyme bay)

The first two examples demonstrate two noun phrases separated by a coordinating conjunction phrase. The third and fourth examples show three noun phrases separated by a comma and a coordinating conjunction phrase. In the fifth example, the [NP hans NP] phrase is a genitive qualifier modifying the [NP börn NP] phrase.

The last example demonstrates a sequence of noun phrases which does not stand for an enumeration.

6.2.1.9 Verb phrases

Our annotation scheme subclassifies verb phrases[3]. A finite verb phrase is labeled as [VP ... VP] and consists of a finite verb optionally followed by a

[3] We use the term *verb phrases* even though our verb phrases are more like *verb clusters* because they can include adverb phrases and more than one verb.

sequence of adverb phrases and supine verbs. Other types of verb phrases are labeled as [VPx ... VPx] where x can have the following values:

- **i**: denoting an infinitive verb phrase
- **b**: denoting a verb phrase which demands a predicate nominative, i.e primarily a verb phrase consisting of the verb "vera" (be).
- **s**: denoting a supine verb phrase
- **p**: denoting a past participle verb phrase
- **g**: denoting a present participle verb phrase

The following are examples of verb phrases:

1. [VP hafði VP] (had)
2. [VP hafði [AdvP stundum AdvP] spjallað VP] (had sometimes talked)
3. [VP hefði [AdvP samstundis AdvP] getað ímyndað VP] (have immediately could imagined)
4. [VPi að halda VPi] (to hold)
5. [VPi að hafa VPi] (to have)
6. [VPb var VPb] (was)
7. [VPb hefur verið VPb] (has been)
8. [VPb reyndist VPb] (turned out to be)
9. [VPs staðið VPs] (stood)
10. [VPs sest VPs] (sit)
11. [VPp orðin VPp] (become)
12. [VPp kominn VPp] (arrived)
13. [VPg æpandi VPg] (screaming)
14. [VPg bölvandi VPg] (cursing)

The first example shows a finite verb phrase consisting of a single finite verb. The second and third examples demonstrate finite verbs followed by an adverb phrase and one or two supine verbs.

Examples no. 4-5 show infinitive verb phrases. Examples no. 6-7 present verb phrases consisting of the verb "be", and example no. 8 includes another verb which demands a nominative complement.

Supine verb phrases are shown in examples no. 9-10. Finally, past and present participle verb phrases are demonstrated in examples no. 11-12 and no. 13-14, respectively.

6.2.1.10 Prepositional phrases

In general, a prepositional phrase [PP ...PP] consists of a preposition (or a MWE phrase which functions as a preposition (MWE_PP)) followed by a sequence of (one or more) noun phrases.

Case government needs to hold between the preposition and the sequence of noun phrases with the exception of an optional sequence of genitive qualifier phrases following or preceding the main noun phrases (see examples below). Furthermore, a prepositional phrase can contain an infinitive verb phrase.

Below we show examples of prepositional phrases:

1. [PP í [NP sögunni NP] PP] (in story)
2. [PP í [NP [AP skuggsælu AP] húsi NP] PP] (in shadowy house)
3. [PP á [NP [APs [AP gula AP] , [AP veðraða AP] APs] múrveggnum NP] PP] (on yellow, weatherworn brick-wall)
4. [PP í [NP sögu NP] [NP fjölskyldunnar NP] PP] (in story family's)
5. [PP [MWE_PP úti við MWE_PP] [NP sjóinn NP] PP] (out by sea)
6. [PP í [NPs [NP haustmyrkri NP] [CP og CP] [NP vetrargnauði NP] NPs] PP] (in autumn-darkness and winter-hiss)
7. [PP [MWE_PP innan um MWE_PP] [NPs [NP [AP gömul AP] húsgögn NP] , [NP [AP latneskar AP] bækur NP] [CP og CP] [NP smyrðlinga NP] NPs] PP]
8. [AP leið AP] [PP á [VPi að sitja VPi] PP] (bored on to sit)

In the first three examples, the prepositional phrases contain a single noun phrase. In the fourth example, a genitive qualifier phrase follows the main noun phrase.

A multiword expression (functioning as a preposition) precedes the noun phrase in example no. 5. Examples no. 6-7 demonstrate a preposition and a multiword expression, respectively, followed by a sequence of noun phrases. The last example shows an infinitive verb phrase following the preposition.

6.2.2 Syntactic functions

Since our constituent structure is flat, functional relations cannot be inferred from hierarchical levels. "Hence, in order to specify, for each relevant phrasal constituent, the function played within the sentence flat structures need to be augmented with explicit functional annotations" (Carroll et al. 1997).

We annotate four different types of syntactic functions: genitive qualifiers, subjects, objects/complements and temporal expressions. We use curly brackets for denoting the beginning and the end of a syntactic function (as carried out, for example, in (Megyesi and Rydin 1999)) and special function tags for labels (*QUAL, *SUBJ, *OBJ/*OBJAP/*OBJNOM/*IOBJ/*COMP, *TIMEX).

6.2.2.1 Genitive qualifiers

A genitive qualifier is a (sequence of) noun phrase(s), marked by the genitive case, which modifies another (usually preceding) noun phrase. The genitive qualifier is marked by {*QUAL ...*QUAL}.

Below, we show examples of such noun phrases:

1. [NP systir NP] {*QUAL [NP hennar NP] *QUAL} (sister hers)
2. [NP börn NP] {*QUAL [NP hans NP] *QUAL} (children his)
3. [NP niðurstöður NP] {*QUAL [NP þessara rannsókna NP] *QUAL} (results this research's)
4. [NP [AP nýkjörinn AP] forseti NP] {*QUAL [NP lýðveldisins NP] *QUAL} (newly-elected president republic's)
5. [PP í [NP sögu NP] {*QUAL [NP fjölskyldunnar NP] *QUAL} PP] (in story family's)

6. [PP á [NP tímum NP] {*QUAL [NPs [NP rútubíla NP] [CP og CP] [NP [AP mikilla AP] mannflutninga NP] NPs] *QUAL} PP]

7. {*QUAL [NP hennar NP] *QUAL} [NP líf NP] (her life)

8. [PP um {*QUAL [NP nokkurra ára NP] *QUAL} [NP skeið NP] PP] (over few year's period)

The first five examples demonstrate a single genitive qualifier noun phrase which modifies a preceding noun phrase. In the sixth example, a sequence of noun phrases functions as the qualifier. The last two examples show qualifier noun phrases preceding the noun phrases that they modify.

6.2.2.2 Subjects

Subjects in Icelandic text are (sequences of) noun phrase(s) appearing, generally, in the nominative case. Exceptions to this rule are noun phrases appearing with special finite verbs which demand subjects in the accusative or dative case. We have compiled a list of these special verbs[4].

Three possible function markers are used for subjects: *{*SUBJ> ... *SUBJ>}*, *{*SUBJ< ... *SUBJ<}* or *{*SUBJ ... *SUBJ}*. The first two tags give information about the relative position of the finite verb. **SUBJ>* means that the verb is positioned to the right of the subject, while **SUBJ<* denotes that the verb is positioned to the left of the subject. Such a relative position indicator is, for example, used in the Constraint Grammar Framework (Karlsson et al. 1995). The last tag is used when it is not clear where the accompanying verb is positioned or when the verb is missing.

Below, we show examples of subject annotations:

1. {*SUBJ> [NP ég NP] *SUBJ>} [VPb var VPb] ... (I was)

2. {*SUBJ> [NP allar óvættir NP] *SUBJ>} [SCP sem SCP] [VP bjuggu VP] ... (all ogresses which)

3. {*SUBJ> [NP systir NP] {*QUAL [NP hennar NP] *QUAL} *SUBJ>} [VPb var VPb] ... (sister hers was)

4. [VPb var VPb] {*SUBJ< [NP ég NP] *SUBJ<} ... (was I)

[4] Thanks to Dr. Jóhannes Gísli Jónsson, University of Iceland, for supplying the original list.

5. [VP kom VP] {*SUBJ< [NP [AP nýkjörinn AP] forseti NP] {*QUAL [NP lýðveldisins NP] *QUAL} *SUBJ<} ... (came newly-elected president republic's)
6. [VP kusu VP] {*SUBJ< [NPs [NP börn NP] {*QUAL [NP hans NP] *QUAL} [CP og CP] [NP niðjar NP] NPs] *SUBJ<} ... (chose children his and descendants)
7. [VP finnst VP] {*SUBJ< [NP þér NP] *SUBJ<} ... (feel-that-way you)
8. {*SUBJ [NP hauststemmning NP] *SUBJ} [PP í [NP Reykjavík NP] PP] (autumn-mood in Reykjavik)

The first three examples show a nominative case subject with the finite verb appearing to the right of it. Examples no. 4-6 demonstrate subjects for which the finite verb is positioned to the left.

Example no. 7 demonstrates a subject in the dative case — the verb "finnast" demands a dative case subject.

Finally, the last example does not have a finite verb, and thus the subject tag does not indicate relative position of the verb.

6.2.2.3 Objects

Our annotation scheme distinguishes between five kinds of verb complements: predicative complements (*{*COMP ... *COMP}*), direct objects (*{*OBJ ... *OBJ}*), indirect objects (*{*IOBJ ... *IOBJ}*), objects of adjectives (*{*OBJAP ... *OBJAP}*), and nominative objects (*{*OBJNOM ... *OBJNOM}*). Moreover, as is the case for subjects, "<" and ">" are used for showing the relative position of the verb.

Predicative complements are complements of verbs which demand a predicate nominative, i.e. primarily the verb *"vera"* (be), and thus appear in the nominative case. Predicative complements can be noun phrases, adjective phrases or past participle verb phrases. Predicative complements can themselves have both objects and predicative complements (see examples below).

Transitive verbs demand direct objects which can appear in any of the oblique cases. Di-transitive verbs demand both direct and indirect objects, for which, typically, the direct object is marked by the accusative case, while the indirect object is marked by the dative case (other case patterns, for direct and indirect objects, are indeed possible, e.g. dative-accusative, accusative-accusative, dative-dative and some patterns with the genitive case).

In some cases, an adjective (phrase) demands an object (see examples below).

The last type of an object, covered by our annotation scheme, is a nominative object of a verb which demands dative case subjects (see examples below).

We assume that parsers using our annotation scheme (e.g. finite-state parsers) do not resolve PP-attachment ambiguities and, thus, our scheme does not extend object noun phrases to include prepositional phrases.

Below, we show examples of object/complement annotation.

1. {*SUBJ> [NP ég NP] *SUBJ>} [VPb var VPb] {*COMP< [AP lítill AP] *COMP<} (I was small)

2. [VPb er VPb] {*SUBJ< [NP ég NP] *SUBJ<} {*COMP< [VPp fædd VPp] [CP og CP] [VPp uppalin VPp] *COMP<} ...(am I born and raised)

3. {*COMP> [AP hávaxinn AP] *COMP>} [VPb er VPb] {*SUBJ< [NP hann NP] *SUBJ<} , {*COMP< [APs [AP vörpulegur AP] , [AP skarpleitur AP] [CP og CP] [AP svipsterkur AP] APs] *COMP<} (tall is he, pretty, sharp-featured and strong-looked)

4. {*SUBJ> [NP Alís NP] *SUBJ>} [VPb var VPb] {*COMP< [VPp orðin VPp] *COMP<} {*COMP< [AP leið AP] *COMP<} (Alís had become bored)

5. {*SUBJ> [NP vagnstjórinn NP] *SUBJ>} [VP sá VP] {*OBJ< [NP mig NP] *OBJ<} (driver saw me)

6. ...[SCP sem SCP] [VP upplýsti VP] {*OBJ< {*QUAL [NP hennar NP] *QUAL} [NP líf NP] *OBJ<} (which enlightened her life)

7. ...[SCP hvorki SCP] [VPi að finna VPi] {*OBJ< [NPs [NP neinar myndir NP] [CP né CP] [NP samtöl NP] NPs] *OBJ<} (neither find any pictures nor conversations)

8. ...{*SUBJ> [NP faðmur NP] {*QUAL [NP hans NP] *QUAL} *SUBJ>} [VP umlykur VP] {*OBJ< [NP [APs [AP lágreist AP] [AP svört AP] APs] húsin NP] *OBJ<}

9. {*OBJ> [NP slíka gagnrýni NP] *OBJ>} [VP læt VP] {*SUBJ< [NP ég NP] *SUBJ<} ... (such criticism let I)
10. {*SUBJ> [NP grundin NP] *SUBJ>} [VPb var VPb] {*COMP< [VPp þakin VPp] *COMP<} {*OBJ< [NP [AP svalri AP] ábreiðu NP] *OBJ<}
11. ... [VPi að segja VPi] {*IOBJ< [NP þér NP] *IOBJ<} {*OBJ< [NP það NP] *OBJ<} (to tell you it)
12. ... [VP hefði [AdvP samstundis AdvP] getað ímyndað VP] {*IOBJ< [NP sér NP] *IOBJ<} {*OBJ< [NPs [NP eitt NP] [CP og CP] [NP annað NP] NPs] *OBJ<}
13. {*SUBJ> [NP ég NP] *SUBJ>} [VPb er VPb] {*COMP< [AP bundin AP] *COMP<} {*OBJAP< [NP Reykjavík NP] *OBJAP<} [NP [AP órjúfanlegum AP] böndum NP] (I am bound Reykjavik ...)
14. {*SUBJ> [NP honum NP] *SUBJ>} [VP fannst VP] {*OBJNOM< [NP hann NP] *OBJNOM<} [VPi sogast VPi] [PP inní PP] (He felt he suck into)

Examples no. 1-4 demonstrate predicative complements, either as adjective phrases or part participle verb phrases. The normal word order is shown in example no. 1, but variants of it are shown in examples no. 2-3. A complement of a complement is shown is example no. 4.

Examples no. 5-8 exhibit objects appearing to the right of the verb (normal word order), whereas example no. 9 shows the object appearing to the left of the verb. Example no. 10, shows a predicative complement which demands a dative object.

Examples no. 11-12 show an annotation for the objects of di-transitive verbs, i.e. indirect and direct objects appearing to the right of a verb phrase.

Finally, examples no. 13-14 show an annotation for an object of an adjective phrase, and for a nominative object of a verb which demands a dative case subject, respectively.

6.2.2.4 Temporal expressions

Temporal expressions in text indicate when something happened, or how long something lasted, or how often something occurs. We use *{*TIMEX ... *TIMEX}* for marking such expressions.

Below, we show examples of temporal expressions.

1. {*TIMEX [NP átta NP] *TIMEX} (eight)
2. {*TIMEX [NP árið 1982 NP] *TIMEX} (year 1982)
3. {*TIMEX [NP þetta kvöld NP] *TIMEX} (this evening)
4. {*TIMEX [NP dag einn NP] *TIMEX} (one day)

6.2.3 The grammar definition corpus

We have constructed a GDC, a corpus consisting of 214 sentences (3738 tokens), whose purpose is to represent the major syntactic constructions in Icelandic, in the following manner. First, we carefully selected the POS tagged sentences from the IFD corpus. Then, we used a preliminary version of our finite-state parser to automatically annotate these sentences. Finally, we checked the annotated sentences with regard to our annotation scheme and hand-corrected all the errors. Tables 6.1 shows the frequency of the various labels for phrases and grammatical functions in our GDC.

The resulting corpus should, along with the annotation scheme itself, provide answers to questions how to analyse a given sentence in Icelandic. Furthermore, this corpus has been used to improve our parser, since we want it to be able to annotate the GDC with high accuracy.

6.3 IceParser

IceParser is an incremental finite-state parser based on the constructive approach (see Section 2.4.2.1). The parser comprises a sequence of finite-state transducers, each of which uses a collection of regular expressions to specify which syntactic patterns are to be recognised. The purpose of each transducer is to add syntactic information into the recognised substrings of the input text.

The input to the parser is POS-tagged sentences. The tags are assumed to be part of the tagset used in the *IFD* corpus, i.e. the tagset used by *IceTagger*[5]. Furthermore, it is assumed that the input file has one sentence

[5]During the development of our parser, we have noticed that the transducers only need to use a part of the features of many of the tags. For example, the following features are not used at all: Gender and number of nominals, declension and degree for adjectives, and voice, person, number and tense for verbs. This fact is a rationale for developing a smaller tagset (as discussed in Section 7.2.6) for the purpose of shallow parsing.

Phrase	Frequency	%	Function	Frequency	%
NP	1308	38.1%	*SUBJ>	260	29.9%
PP	476	13.9%	*OBJ<	151	17.4%
AP	291	8.5%	*COMP<	140	16.1%
VP	280	8.2%	*QUAL	103	11.9%
AdvP	231	6.7%	*SUBJ<	100	11.5%
CP	183	5.3%	*SUBJ	37	4.3%
VPb	178	5.2%	*TIMEX	24	2.8%
SCP	113	3.3%	*COMP>	16	1.8%
VPi	103	3.0%	*COMP	11	1.3%
NPs	69	2.0%	*IOBJ<	8	0.9%
VPp	58	1.7%	*OBJ>	7	0.8%
MWE_CP	43	1.3%	*OBJAP>	5	0.6%
MWE_PP	35	1.0%	*OBJAP<	5	0.6%
APs	24	0.7%	*OBJNOM<	2	0.2%
MWE_AdvP	21	0.6%			
VPs	9	0.3%			
InjP	5	0.1%			
VPg	2	0.1%			
MWE_AP	1	0.0%			
Total:	3430	100.0%		869	100.0%

Table 6.1: The frequency of the various labels in the GDC

in each line. The output of the parser thus consists of the POS-tagged sentences with added syntactic information.

IceParser is designed to produce annotations according to the annotation scheme described in Section 6.2. The parser consists of two main components: the phrase structure module and the syntactic functions module. "The thought behind 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 total, *IceParser* consists of 22 finite-state transducers.

The purpose of the phrase structure module is to add brackets and labels to input sentences to indicate phrase structure and linguistic information. The output of one transducer serves as the input to the following transducers in the sequence. The syntactic annotation is performed in a bottom-up

fashion, i.e. deepest constituents are analysed first. For example, adverb phrases are marked before adjective phrases, which are in turn marked before noun phrases.

Both simple phrase structures and complex structures are recognised. Since the parser is based on finite-state machines, each phrase structure does not contain a structure of the same type. Complex structures contain other structures, whereas simple structures do not.

The purpose of the syntactic functions module is to add functional tags to denote grammatical functions. The input to the first transducer in this module is the output of the last transducer in the phrase structure module, i.e. it is assumed that the syntactic functions module receives text that has been annotated with constituent structure. As in the phrase structure module, the output of one transducer serves as the input to the following transducers in the sequence.

As discussed in Chapter 3, feature agreement (in gender, number and case) in Icelandic, between a noun and its modifiers, indicates which words belong together and form constituents. Nevertheless, our parser makes minimal use of feature agreement when annotating constituents. Instead, it employs mainly word class and word order information when forming phrases. The case feature is, however, used extensively when assigning grammatical functions.

The reason for this is that we want our parser to be utilised as a grammar correction tool, among other things. If the parser uses feature agreement to a great extent to mark phrases then it will not be possible for a grammar correction tool to point out feature agreement errors inside phrases. This is because the corresponding words would not have been recognised as one phrase by the parser, due to the lack of feature agreement! Indeed, evaluation shows (see Chapter 8) that by using mainly word class and word order information, the accuracy of our parser is remarkably good.

6.3.1 Implementation

The parser is implemented in Java and the lexical analyser generator tool JFlex (http://jflex.de/). Each transducer is written in a separate file, which is compiled into Java code using JFlex. The resulting Java code is a deterministic finite-state automaton (DFA), along with actions to execute for each recognised pattern. The reason for not using a tool like the XFST for implementation is that *IceParser* is part of the *IceNLP* toolkit, all of which

is implemented in Java. Writing the parser in Java has enabled us to easily integrate the parser with the other Java components of the *IceNLP* tool, e.g. *IceTagger*. Furthermore, by having full control of the source, we have been able to build an optimised version of *IceParser* (as discussed in Section 8.2).

The architecture of *IceParser*, along with the name of each transducer, is shown in Figure 6.1. The first transducer in the phrase structure module is called *Phrase_MWE*. It reads an input file consisting of POS tagged sentences and marks MWEs by using patterns written as regular expressions (see Section 6.3.2). The next transducers in the sequence, *Phrase_MWEP1*, takes as input the output generated from the previous transducer and produces output ready to be read by the next transducer in the sequence.

The architecture of the syntactic functions module is similar. The first transducer, *Func_TIMEX*, recognises temporal expressions. It reads an input file which has been annotated with constituent structure, i.e. the output file generated by the last transducer in the phrase structure module (*Clean1*). The output of the *Func_TIMEX* transducer is then read by the next transducer, *Func_QUAL*, in the sequence, etc.

Table 6.2 very briefly describes the purpose of each transducer – please refer to Appendix C for a thorough description of all the transducers of the parser.

Name	Purpose
Phrase_MWE	Marks MWEs consisting of common bigrams and trigrams.
Phrase_MWEP1	Marks MWEs consisting of specific <preposition, adverb> pairs.
Phrase_MWEP2	Marks MWEs consisting of specific <adverb, preposition> pairs.
Phrase_AdvP	Marks adverb phrases, conjunction phrases 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 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 and other special types of objects.
Func_OBJ3	Marks dative objects of complement adjective phrases.
Func_SUBJ2	Marks "stand-alone" nominative noun phrases.
Clean2	Clean up.

Table 6.2: A brief description of all the transducers.

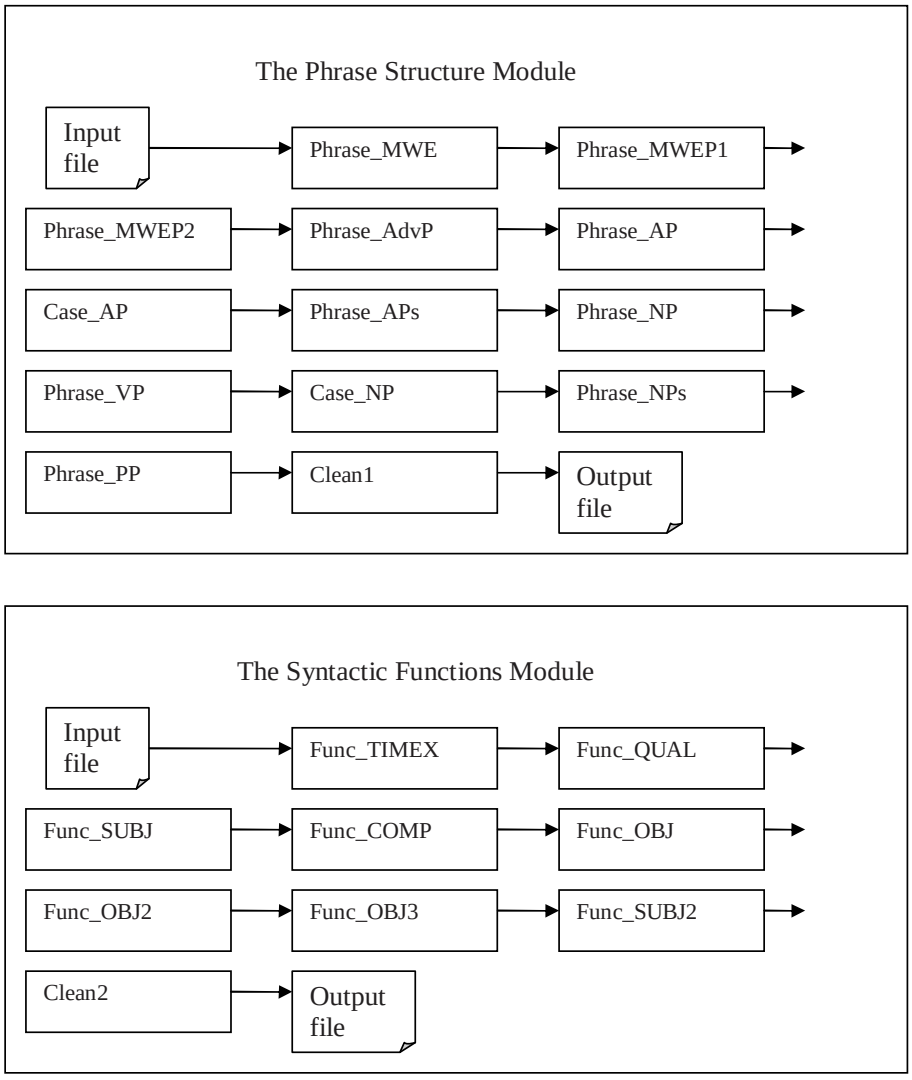

Figure 6.1: The architecture of *IceParser*

6.3.2 Rules and actions

The patterns recognised by each transducer are written using the regular expressions language of JFlex. Table 6.3 shows the main regular expression operators supported by JFlex (borrowed from (Klein 2005)).

Operator	Usage	Descripton
Union	$a\|b$	Matches all input matched by a or by b.
Concatenation	ab	Matches all input matched by a followed by the input matched by b.
Kleene closure	a^*	Matches zero or more repetitions of the input matched by a.
Iteration	a^+	Matches one or more repetitions of the input matched by a.
Option	$a?$	Matches the empty input or the input matched by a.
Upto	$\sim a$	Matches everything up to (and including) the first occurrence of the input matched by a.
Repeat	$a\{n\}$	Equivalent to n times the concatenation of a.

Table 6.3: The main regular expression operators supported by JFlex.

The rules section of a JFlex specification contains regular expressions and actions (Java code) that are executed when the tool matches the associated regular expression. As an example of the rule (and action) format, 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}  { System.out.print(Open+yytext()+Close);}
```

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

Two regular definitions[6], *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*. The *AdverbTag* is a name defined in a separate file *phraseDef.txt*, 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* (see the description of the Icelandic tagset in Appendix B) and postfixed with one or more white spaces.

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

The above code is a simple illustration of the format of the rules and actions in the source files for the transducers. Please refer to Appendix C for a thorough description of the rules and actions used by *IceParser*.

6.3.3 An illustrative example

In this section, we demonstrate the output of the various transducers of *IceParser* when parsing the following sentence from our GDC (shown with POS tags):

```
við fp1fn mættumst sfm1fþ á aþ gangstéttinni nveþg , ,
we        met             on   pavement-the        ,

heilsuðumst sfm1fþ   og c tókum sfg1fþ
greeted (each other) and  took

tal nheo saman aa eins aa og c  gamlir lkfnsf kunningjar nkfn . .
talk     together like          old           pals            .
```

[6] 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.

The first transducer in the phrase structure module, *Phrase_ MWE*, annotates the bigram *eins aa og c* as a MWE functioning as a conjunction (because the transducer includes a pattern matching this exact MWE). The resulting output is:

við fp1fn mættumst sfm1fþ á aþ gangstéttinni nveþg , , heilsuðumst sfm1fþ og c tókum sfg1fþ tal nheo saman aa [MWE_ CP eins aa og c MWE_ CP] gamlir lkfnsf kunningjar nkfn . .

The next transducer which adds information into the text is *Phrase_ AdvP*. It annotates *og c* as a conjunction phrase and *saman aa* as an adverb phrase, resulting in:

við fp1fn mættumst sfm1fþ á aþ gangstéttinni nveþg , , heilsuðumst sfm1fþ [CP og c CP] tókum sfg1fþ tal nheo [AdvP saman aa AdvP] [MWE_ CP eins aa og c MWE_ CP] gamlir lkfnsf kunningjar nkfn . .

The *Phrase_ AP* transducer marks the adjective *gamlir lkfnsf*, resulting in:

við fp1fn mættumst sfm1fþ á aþ gangstéttinni nveþg , , heilsuðumst sfm1fþ [CP og c CP] tókum sfg1fþ tal nheo [AdvP saman aa AdvP] [MWE_ CP eins aa og c MWE_ CP] [AP gamlir lkfnsf AP] kunningjar nkfn . .

The *Case_ AP* transducer adds case information to the adjective phrases (appending the letter *n* denoting nominative case to the opening label of the phrase), resulting in:

við fp1fn mættumst sfm1fþ á aþ gangstéttinni nveþg , , heilsuðumst sfm1fþ [CP og c CP] tókum sfg1fþ tal nheo [AdvP saman aa AdvP] [MWE_ CP eins aa og c MWE_ CP] [APn gamlir lkfnsf AP] kunningjar nkfn . .

This illustrative example does not include a sequence of adjective phrases and therefore the application of the *Phrase_ APs* transducer does not change the string. The next transducer, *Phrase_ NP*, marks the noun phrases *við fp1fn*, *gangstéttinni nveþg*, *tal nheo*, and *[APn gamlir lkfnsf AP] kunningjar nkfn*, resulting in:

[NP við fp1fn NP] mættumst sfm1fþ á aþ [NP gangstéttinni nveþg NP] , , heilsuðumst sfm1fþ [CP og c CP] tókum sfg1fþ [NP tal nheo NP] [AdvP saman aa AdvP] [MWE_CP eins aa og c MWE_CP] [NP [APn gamlir lkfnsf AP] kunningjar nkfn NP] . .

The *Phrase_VP* transducer annotates the verbs *mættumst sfm1fþ*, *heilsuðumst sfm1fþ* and *tókum sfg1fþ*, resulting in:

[NP við fp1fn NP] [VP mættumst sfm1fþ VP] á aþ [NP gangstéttinni nveþg NP] , , [VP heilsuðumst sfm1fþ VP] [CP og c CP] [VP tókum sfg1fþ VP] [NP tal nheo NP] [AdvP saman aa AdvP] [MWE_CP eins aa og c MWE_CP] [NP [APn gamlir lkfnsf AP] kunningjar nkfn NP] . .

The *Case_NP* transducer adds case information to the noun phrases (*n* for nominative, *a* for accusative), resulting in:

[NPn við fp1fn NP] [VP mættumst sfm1fþ VP] á aþ [NPd gangstéttinni nveþg NP] , , [VP heilsuðumst sfm1fþ VP] [CP og c CP] [VP tókum sfg1fþ VP] [NPa tal nheo NP] [AdvP saman aa AdvP] [MWE_CP eins aa og c MWE_CP] [NPn [APn gamlir lkfnsf AP] kunningjar nkfn NP] . .

This illustrative example does not include a sequence of noun phrases and therefore the application of the *Phrase_NPs* transducer does not change the string. The next transducer, *Phrase_PP*, marks the prepositional phrase *á aþ [NPd gangstéttinni nveþg NP]*, resulting in:

[NPn við fp1fn NP] [VP mættumst sfm1fþ VP] [PP á aþ [NPd gangstéttinni nveþg NP] PP] , , [VP heilsuðumst sfm1fþ VP] [CP og c CP] [VP tókum sfg1fþ VP] [NPa tal nheo NP] [AdvP saman aa AdvP] [MWE_CP eins aa og c MWE_CP] [NPn [APn gamlir lkfnsf AP] kunningjar nkfn NP] . .

Finally, the application of the *Clean1* transducer does not affect the final output string generated by the phrase structure module.

The *Func_SUBJ* transducer is the first transducer in the syntactic functions module which adds new information to the input string. It recognises that the noun phrase *[NPn við fp1fn NP]* is a subject (because one of the patterns of the transducer matches an nominative noun phrase preceding a

verb phrase), resulting in:

*{*SUBJ> [NPn við fp1fn NP] *SUBJ>} [VP mættumst sfm1fþ VP] [PP á aþ [NPd gangstéttinni nveþg NP] PP] , , [VP heilsuðumst sfm1fþ VP] [CP og c CP] [VP tókum sfg1fþ VP] [NPa tal nheo NP] [AdvP saman aa AdvP] [MWE_CP eins aa og c MWE_CP] [NPn [APn gamlir lkfnsf AP] kunningjar nkfn NP] . .*

The input sentence does not have a verb complement and therefore the *Func_COMP* transducer does not add any new information. The *Func_OBJ* transducer does, however, recognise the noun phrase *[NPa tal nheo NP]* as an object (because the noun phrase is in the accusative case and follows a verb phrase). The resulting output is:

*{*SUBJ> [NPn við fp1fn NP] *SUBJ>} [VP mættumst sfm1fþ VP] [PP á aþ [NPd gangstéttinni nveþg NP] PP] , , [VP heilsuðumst sfm1fþ VP] [CP og c CP] [VP tókum sfg1fþ VP] {*OBJ< [NPa tal nheo NP] *OBJ<} [AdvP saman aa AdvP] [MWE_CP eins aa og c MWE_CP] [NPn [APn gamlir lkfnsf AP] kunningjar nkfn NP] . .*

Func_SUBJ2 is the next transducer in the sequence which adds new information to the string. It recognises the noun phrase *[NPn [APn gamlir lkfnsf AP] kunningjar nkfn NP]* as a subject (because it is in the nominative case and has not yet been marked with a syntactic function).

The resulting output is:

*{*SUBJ> [NPn við fp1fn NP] *SUBJ>} [VP mættumst sfm1fþ VP] [PP á aþ [NPd gangstéttinni nveþg NP] PP] , , [VP heilsuðumst sfm1fþ VP] [CP og c CP] [VP tókum sfg1fþ VP] {*OBJ< [NPa tal nheo NP] *OBJ<} [AdvP saman aa AdvP] [MWE_CP eins aa og c MWE_CP] {*SUBJ [NPn [APn gamlir lkfnsf AP] kunningjar nkfn NP] *SUBJ} . .*

Note that for this last subject there is no accompanying verb and therefore no relative position marker ("<" or ">") is used. Finally, the *Clean2* transducer "cleans up" the string, e.g. removes the case information attached to *NP* and *AP* labels, and removes unnecessary extra spaces. The final output is thus:

*{*SUBJ> [NP við fp1fn NP] *SUBJ>} [VP mættumst sfm1fþ VP] [PP á aþ [NP gangstéttinni nveþg NP] PP] , , [VP heilsuðumst sfm1fþ VP] [CP og c CP] [VP tókum sfg1fþ VP] {*OBJ< [NP tal nheo NP] *OBJ<} [AdvP saman aa AdvP] [MWE_ CP eins aa og c MWE_ CP] {*SUBJ [NP [AP gamlir lkfnsf AP] kunningjar nkfn NP] *SUBJ} . .*

Part III

Evaluation

Chapter 7

Tagging Icelandic Text

7.1 Introduction

7.1.1 Previous work

The earliest tagging results for Icelandic text were published by Briem (1989). The tagger used was a mixture of a linguistic rule-based tagger and a probabilistic tagger. Each linguistic rule had points associated with it and the points were allocated to a rule on the basis of frequency information, derived from a hand-tagged corpus of 54,000 tokens. For each possible tag sequence the program calculated the total number of points given by the rules when they were satisfied. The tag sequence with the highest number of points was chosen as "the best sequence". In order to prevent an enormous amount of possible tag sequences the tagger worked with only a part of each sentence at a time.

The accuracy of the tagger was reported as 70% but 15% of the tokens were not analysed. The author later improved the program, by using frequency information from the whole *IFD* corpus (described in Section 4.1.1) and obtained an accuracy of a little less than 90% (personal communication).

In 2002-2004, the Institute of Lexicography (IL) at The University of Iceland performed an Icelandic tagging experiment (ITE) (Helgadóttir 2004) using three state-of-the-art data-driven taggers: *fnTBL* (Ngai and Florian 2001), *MXPOST* (Ratnaparkhi 1996) and *TnT* (Brants 2000). As discussed in Section 2.3.4, the *fnTBL* tagger is a fast implementation of transformation-based error-driven learning, the *MXPOST* tagger uses a maximum entropy method and the *TnT* tagger is a statistical tagger based on a second order

Markov model.

The *IFD* corpus was utilised for training and testing in the ITE, and ten-fold cross validation was used. Each test corpus has 59,030 tokens (3,691 sentences), on average, and the average unknown word ratio is 6.84%. The highest accuracy, 90.36%, was obtained by the *TnT* tagger — the average results for the three taggers can be seen in Table 7.1.

By using a weighted voting scheme, in which each tagger was weighted with its accuracy (this is equivalent to simple voting when using three taggers), the total accuracy increased to 91.54%. When using backup dictionaries for *TnT* and *fnTBL* (to decrease the ratio of unknown words), simple voting on a simplified tagset and applying linguistically motivated rules in the combination of the taggers, the accuracy increased to 93.65%.

7.1.1.1 Discussion

The tagging accuracy of 90.36% in the *ITE*, obtained by the best performing single tagger, is considerably lower than the one achieved for related languages, e.g. Swedish where 93.55% accuracy was obtained in an experiment using the same taggers (see Table 7.2), a tagset consisting of 139 tags, and a training corpus of only 100k tokens (Megyesi 2002). This difference in accuracy can be explained by the large Icelandic tagset[1], as well as by the fact that Icelandic is morphologically considerably more complicated than Swedish.

We have previously shown (see Table 4.1) that the ambiguity rate of Icelandic text, computed using the *IFD* corpus is 2.74, compared to, for example, 2.05 for Swedish text as determined by a dictionary used in a CG Framework (Birn 1998). The ambiguity rate seems, however, to be similar to ambiguity rate of English text (2.82) using the Lancaster-Oslo/Bergen corpus (170 tags) (van Halteren et al. 2001). On the other hand, using another frequently used criterion for ambiguity of a language, the ratio of ambiguous tokens in text to the total number of tokens, we found 59.7% of the tokens in the *IFD* corpus (again see Table 4.1) to be ambiguous. Compared to English,

[1]Intuitively, a larger tagset should result in lower tagging accuracy, since, for a larger tagset, the tagger simply has more tags to choose from for each word. In a tagging experiment on Dutch text, using a tagset of 341 tags, 92.06% accuracy was achieved by the *TnT* tagger (van Halteren et al. 2001). In an experiment on tagging Czech text, using a very large tagset of 1,171 tags and a trigram tagger, an accuracy of only 81.1% was obtained (Hajič and Hladká 1998).

Accuracy/Tagger	fnTBL	MXPOST	TnT
Unknown word accuracy	54.03%	62.50%	71.60%
Known word accuracy	91.36%	91.04%	91.74%
Total accuracy	88.80%	89.08%	90.36%

Table 7.1: Average tagging accuracy in the Icelandic Tagging Experiment (ITE)

this is a much higher ratio where, for example, 35% of the word tokens in the Brown corpus were found to be ambiguous (Kupiec 1992).

An apparent difference between the Swedish experiment cited above and the *ITE* is a considerable difference in the tagging accuracy of unknown words (compare tables 7.1 and 7.2). This difference seems to indicate that it is more difficult to guess tags for unknown words in Icelandic compared to Swedish, and indeed, as previously mentioned, Icelandic is morphologically more complex than Swedish.

It is worth mentioning that two standard reported measures of ambiguity, the ratio of ambiguous tokens and the average number of tags assigned to tokens (i.e. ambiguity rate), can not always been used as an indication of how hard the task for a tagger actually is. This is, for example, pointed out by Oravecz and Dienes (2002) who show that even though the two measures for Hungarian are considerable lower than for English, the former language is actually more difficult to tag. This is due to the inflectional nature of Hungarian which results in a much larger number of word types (more than twice) in a Hungarian corpus when compared to an English corpus of similar size. Consequently, for lexicalised taggers like trigram taggers, Hungarian is more difficult to tag.

For the case of Icelandic vs. Swedish, the ratio of word types to word tokens is about the same (see Table 4.1 and (Megyesi 2002)), and hence the difference in tagging accuracy must be explained with other factors like the difference in tagset size, ambiguity rate, ratio of ambiguous tokens (and different corpora used for training and testing).

The difference in tagging accuracy of Icelandic text between the three data-driven taggers can by and large be attributed to the difference in accuracy when tagging unknown words. The tagging accuracy of *fnTBL* for unknown words is relatively poor, about 25% less than the corresponding accuracy

Accuracy/Tagger	fnTBL	MXPOST	TnT
Unknown word accuracy	58.52%	78.85%	82.29%
Known word accuracy	94.35%	93.34%	95.50%
Total accuracy	89.06%	91.20%	93.55%

Table 7.2: Average tagging accuracy in a Swedish tagging experiment (Megyesi 2002)

of *TnT*. On the other hand, since *fnTBL* achieves relatively high accuracy on known words, one can assume that with a better unknown word guessing module the total accuracy of *fnTBL* can be improved. Indeed, in Section 7.3, we will show how to substantially improve *fnTBL's* unknown word tagging accuracy, resulting in total accuracy of 90.15%.

The relatively poor tagging accuracy for unknown words in *fnTBL* can probably be explained by the following. Recall that the most frequent tag for each word is used for initial tagging. When tagging English text, a transformation-based tagger typically uses a singular proper noun tag for capitalised words, and a singular common noun tag for other words, for initial tagging of unknown words. This works well since only one tag in an English tagset is needed for singular proper nouns and singular common nouns, respectively, and, thus, it is to be expected that this initial assignment produces the most likely tag for each unknown word. In Icelandic, however, due to gender and case, there is more than one possible tag for the singular common nouns and proper nouns. Hence, by choosing one particular common noun tag and one particular proper noun tag for initial tagging of all unknown words, it is highly probable that most of the time the chosen initial tag will not be the most likely tag for a given unknown word.

7.2 Evaluation of IceTagger

In this section, we evaluate the tagging accuracy of *IceTagger*. First, we evaluate the unknown word guesser, *IceMorphy*. Then, we evaluate the performance of the heuristics used by *IceTagger*, and, finally, we present tagging accuracy results for the tagger as a whole.

7.2.1 The unknown word guesser

For the purpose of evaluating the performance of our unknown word guesser, *IceMorphy*, we hand-tagged 400 unknown words (i.e., we generated the true tag profile for these words), randomly extracted from the first test corpus of the *IFD* corpus. The number of relevant (correct) tags was 1194, but *IceMorphy* generated 1554 tags, which divided in the following manner: 555 (35.71%) common noun tags, 266 (17.12%) proper noun tags, 553 (35.59%) adjective tags, 170 (10.94%) verb tags, and 10 (0.64%) tags belonging to other word classes.

Table 7.3 shows precision, recall and F-measure for the main word classes and for all tags as a group. Precision, recall and F-measure are defined in the following manner:

$$precision = \frac{\#\text{ of relevant generated tags}}{\#\text{ of generated tags}} \tag{7.1}$$

$$recall = \frac{\#\text{ of relevant generated tags}}{\#\text{ of relevant tags}} \tag{7.2}$$

$$F\text{-}measure = \frac{2 * precision * recall}{precision + recall} \tag{7.3}$$

For the purpose of tagging, high recall is more important than high precision. It is imperative that an unknown word guesser produces as many relevant tags as possible for a tagger to disambiguate. If a relevant tag is missing, a tagger may not be able to disambiguate correctly.

	Common nouns	Proper nouns	Adjectives	Verbs	All words
Precision	64.32%	18.80%	72.73%	52.35%	58.17%
Recall	83.80%	58.14%	71.43%	82.41%	75.71%
F-measure	72.78%	28.41%	71.88%	64.03%	65.79%

Table 7.3: Accuracy for the given word classes when using *IceMorphy* to guess tags for 400 randomly selected words

Let us assume that the figures in Table 7.3 represent the precision and

	Morphological analysis	Compound analysis	Ending analysis	Default analysis
Precision	60.84%	86.22%	21.00%	53.85%
Recall	78.36%	79.89%	52.76%	46.67%
F-measure	68.50%	82.93%	30.04%	50.00%

Table 7.4: Accuracy for the different modules of *IceMorphy* when guessing tags for 400 randomly selected words

recall figures for the whole population of unknown words analysed by *IceMorphy*. Based on this assumption, one might reason that the recall figures place an approximate upper bound on the tagging accuracy obtainable by *IceTagger* for unknown words. However, this is not true because even though a tag profile may be incomplete for a particular word it may have little impact on the tagging accuracy if the omitted tags are very rare.

The low precision and recall for proper nouns can be explained by the following. The last character of the tag for proper nouns denotes named entity information, i.e. a person name, place name or any other name. Since *IceMorphy* does not include a named entity recogniser, it has difficulty guessing the correct tag for proper names. When the last character of proper nouns is ignored, precision and recall for proper nouns increases to 31.20% and 96.51%, respectively.

Table 7.4 shows precision, recall and F-measure for the different modules of *IceMorphy* (see Section 5.2). The figures in this table show that morphological analysis and compound analysis are more accurate than ending analysis (as discussed in Section 2.3.6.1). On the other hand, note that ending analysis is only carried out if morphological/compound analysis is unsuccessful.

No morphological analyser for the Icelandic language has previously been published and, thus, we can not compare our figures to results published for Icelandic.[2] Additionally, it is difficult to do comparison across languages because of different levels of morphological complexity and, hence, different tagsets. For the sake of doing one comparison, the morphological classifier of

[2]The morphological analyser "Púki" is a spelling and grammar tool developed by a private Icelandic software company. The software is proprietary, intended to be used with Microsoft Office 2003, and accuracy figures for the analyser have not been published.

unknown German nouns described by Nakov et al. (2003) achieved 82%-89% recall (depending on the test corpora). The purpose of the German analyser was to guess morphological classes using a comprehensive stem dictionary, as opposed to predicting individual tags using a dictionary derived from a tagged corpus, as is the case for *IceMorphy*. Since using a more comprehensive dictionary will most certainly increase *IceMorphy's* accuracy, we believe our guesser obtains good results.

7.2.2 The heuristics

In Section 7.2.3, we will show that the heuristics have a large impact on the overall tagging accuracy of *IceTagger*. In this section, however, we evaluate the heuristics *per se*, i.e. the accuracy of the syntactic and functional (SynFunc) tagging, which the heuristics base their disambiguation process on.

We built a *gold standard* by randomly selecting 150 sentences from the first test corpus of the *IFD* corpus and hand-tagged these sentences with SynFun tags, i.e. *PP* tags and *SUBJ*, *VERB* and *OBJ* tags. The sentences contain a total of 2,868 tokens, i.e. 19.1 tokens per sentence, on the average. During hand-tagging, 1,691 (59%) tokens received a SynFun tag.

We then ran *IceTagger* on the 150 sentences and computed precision and recall for the SynFun tags generated by the tagger. The POS tagging accuracy of *IceTagger* for these sentences was 92.29%, and the ratio of unknown words was 8.26%.

Table 7.5 shows how the 1,691 tokens divide between the four SynFun tags, both in the gold standard and in *IceTagger*. Not surprisingly, the number of *PP* tags is highest because each word in a PP (with the exception of an adverb) is tagged. Furthermore, *VERB* tags outnumber *SUBJ* and *OBJ* tags because a verb(s) occurs in almost every sentence. More *SUBJ* tags than *OBJ* tags are found which can be explained by the fact that not all verbs are transitive (i.e. not all verbs need an object), but a subject is, generally, needed.

Table 7.6 shows precision, recall, and F-measure for the different tag types, guessed by the heuristics. The table shows much higher F-measure for *VERB* and *PP* tags compared to *SUBJ* and *OBJ* tags. This is to be expected because guessing the former is much easier than guessing the latter. As explained in Section 5.3.5.2, a token receives a *VERB* functional tag if the first POS tag, in its (locally disambiguated) tag list, is a verb tag. Similarly,

Tag	Gold standard	Generated by *IceTagger*
SUBJ	265 (15.7%)	254 (15.4%)
VERB	425 (25.1%)	423 (25.6%)
OBJ	216 (12.8%)	219 (13.3%)
PP	785 (46.4%)	754 (45.7%)
Total	1,691 (100%)	1,650 (100%)

Table 7.5: Partion of SynFun tag types

Tag	Precision	Recall	F-measure
SUBJ	85.43%	81.89%	83.62%
VERB	94.56%	94.12%	94.34%
OBJ	72.60%	73.61%	73.10%
PP	97.61%	93.76%	95.65%

Table 7.6: Precision, recall and F-measure for SynFun tag types, guessed by the heuristics

Tag	Precision	Recall	F-measure
SUBJ	86.10%	84.15%	85.11%
VERB	94.84%	95.06%	94.95%
OBJ	75.12%	74.07%	74.59%
PP	97.24%	94.27%	95.73%

Table 7.7: Precision, recall and F-measure for SynFun tag types, guessed by the heuristics, when using a closed vocabulary

a preposition candidate is easy too guess and the accompanying PP words are just those nominals having the same case as the preposition. Guessing the functional *SUBJ* and *OBJ* tags is, however, more difficult because the correct guess is not only dependent on the word class, but also on word order and verb subcategorisation information.

Recall that 8.26% of the tokens, behind the figures in Table 7.6, were unknown to the tagger. As expected, the accuracy improves by including the unknown words in the dictionary (i.e. using a closed vocabulary), as can been seen by comparing tables 7.6 and 7.7 (the POS tagging accuracy using a closed vocabulary is 94.25%).

7.2.2.1 Discussion

There are various different causes for errors in the *SUBJ* and *OBJ* tagging. One source of error is the lack of verb subcategorisation information in *IceTagger*. For example, in the sentence "*þarna svelgdist ykkur á bjórnum*" (*there quaff you on beer*) the verb "*svelgdist*" demands a dative subject (but not the usual nominative subject) and, hence, the pronoun "*ykkur*" should be tagged with a *SUBJ* tag, but not an *OBJ* tag.

Another problematic situation occurs when an (implicit) subject is missing from a clause, like "*en verða varir ...*" (*but become aware ...*). Since the subject "*þeir*" (*they*) is missing, the adjective "*varir*" is incorrectly tagged as a subject instead of a verb complement.

Table 7.6 shows substantially higher F-measure for *SUBJ* vs. *OBJ*. We have noticed that PPs are responsible for many of the *OBJ* errors (and, indeed, some of the *SUBJ* errors as well). In the sentence "*hann heyrði með öðru eyranu hljóðin*" (*he heard with one ear sounds*), the noun "*hljóðin*" is a direct object of the verb "*heyrði*", but the PP "*með öðru eyranu*" lies between the verb and the object. The heuristic described in Section 5.3.5.5 does not handle such intervening PPs. Furthermore, in some cases, *IceTagger* tags OBJs as VERBs, due to lack of an appropriate local disambiguation rule.

Our error analysis implies that the accuracy of *SUBJ*/*OBJ* tagging may be improved by the following. First, by adding more thorough verb subcategorisation information to *IceTagger*. Secondly, by "stepping over" intervening PPs, between the verb and the corresponding *SUBJ* or/and *OBJ*, when searching for subjects and objects. Lastly, by writing more local rules, thus eliminating more inappropriate tags before the heuristics are applied. Consequently, improving the accuracy of *SUBJ*/*OBJ* tagging will most probably

increase the POS tagging accuracy of *IceTagger*.

It would be interesting to compare the figures for the functional *SUBJ* and *OBJ* tags with corresponding evaluation figures produced by a parser for Icelandic text. Unfortunately, no such figures are available[3]. Several results on tagging grammatical functions have, however, been published for related languages.

A recent study on grammatical function assignment for German (using memory-based learning from a corpus annotated with grammatical functions tags), showed F-measure as 87.23%, 78.60% and 75.32%, for subjects, accusative objects and verb complements, respectively (Kouchnir 2004) (recall that our figures for *OBJ* tags include both direct objects and verb complements). In another German study (using finite-state cascades to annotate grammatical functions on top of a shallow constituent structure), the corresponding F-measures were 90.77%, 81.86% and 79.61%, respectively (Müller 2004).

Since both these methods are based on parsing, higher scores are to be expected in comparison to our (non-parsing) heuristics. Note as well, that the above methods perform grammatical tagging using fully disambiguated POS tags, whereas our grammatical tagging component is used to facilitate the POS tag disambiguation. In our opinion, this comparison shows that the accuracy of our heuristics for tagging subjects and objects of verbs is relatively high. Moreover, improving the accuracy of these heuristics is possible, as discussed above.

7.2.3 Accuracy of IceTagger

In order to make a fair comparison between *IceTagger* and the data-driven taggers, we used exactly the same training and test corpora as were used in the ITE. Recall that the available hand-tagged corpus is the *IFD* corpus consisting of about 590,000 tokens. Pairs of 10 training corpora (each containing 90% of the *IFD* corpus) and 10 test corpora (each containing 10% of the *IFD* corpus) were constructed in the ITE. We used the first nine of these test corpora for evaluation — the tenth test corpus was set aside and used as the development corpus.

For each test corpus the corresponding training corpus was used to deduce part of the dictionaries used by *IceTagger* — the main dictionary stating word

[3]Indeed, only one parser for the Icelandic language currently exists. It is a parser based on HPSG, developed by a private Icelandic software company. (Note that our parser, *IceParser*, was developed after the evaluation of the heuristics was carried out.)

forms and allowable tags (55,600 word forms, on the average), the dictionary for phrasal verb recognition, the dictionary for verb-preposition pairs and the dictionary for verb-object case governance. The same holds for *IceMorphy*, i.e. a part of its dictionaries (the main dictionary and list of endings) were also deduced from the corresponding training corpus. Thus, unknown words in tests of the *TnT* tagger are also unknown in tests of *IceTagger*, with one exception.

IceTagger uses an additional base dictionary which includes words of the closed word classes: pronouns, prepositions, conjunctions and about 120 irregular verbs. Since these words are, indeed, very common and are therefore in most cases already in the dictionary derived from a training corpus, the ratio of unknown words, when testing *IceTagger*, is only a fraction lower than the corresponding ratio in the ITE (6.79% vs. 6.84% on the average). Table 7.8 shows statistics for the nine test corpora used in our experiment.

Before we present the results, let us first discuss baseline accuracy figures for tagging Icelandic text using the given tagset. By baseline accuracy, we mean the lower bound on the accuracy that our tagger should achieve. A naive baseline tagger can be constructed by always assigning each known word its most frequent tag, and selecting the most frequently occurring tag for unknown common nouns and unknown proper nouns, respectively. The average baseline tagging accuracy for the nine test corpora using this method is 76.27%. This figure is considerably lower than the baseline accuracy of 80.75% for Swedish (Megyesi 2002). Using similar methods for known words and assigning unknown words the tag most common for words ending in the same three letters, a baseline tagging accuracy of around 92% has been reported for English (Brill 1992).

The tagging accuracy of *IceTagger* for the nine test corpora can be seen in Table 7.9. The average accuracy for unknown words, known words and all words is 75.09%, 92.74% and 91.54%, respectively[4]. As mentioned above,

[4]Recall (from Section 5.3.4) that the tags in the tag profile for the each word are checked in decreasing order of frequency, during the application of local rules. When this thesis was revised, we evaluated *IceTagger* by processing the tags for each word in ascending order of frequency. In that case, the average accuracy for unknown words, known words and all words is 75.15%, 92.79% and 91.60%, respectively. Higher accuracy was obtained for each test corpus, and for seven out of the nine test corpora the difference is statistically significant ($\alpha < 0.05$, using McNemar's chi-squared goodness-of-fit test as described by Dietterich (1998)). A probable explanation for a slightly higher accuracy when processing the tags in the ascending order of frequency is the following. In some cases, a local rule in *IceTagger* removes a legitimate (and at the same time a frequent) tag t for a given word

						Baseline accuracy		
Test corpus	# of tokens	# of sentences	unkn. ratio	ambig. words	ambig. rate	unkn. words	known words	all words
01	59,169	3,503	7.54%	60.56%	2.79	4.77%	81.46%	75.43%
02	58,967	3,601	6.74%	60.30%	2.79	3.63%	81.60%	76.06%
03	59,077	3,541	6.81%	60.45%	2.78	4.27%	81.91%	76.35%
04	59,067	3,776	6.64%	60.31%	2.76	4.62%	82.34%	76.85%
05	59,076	3,861	6.46%	60.37%	2.74	4.53%	82.14%	76.83%
06	59,136	3,748	6.68%	60.20%	2.74	3.99%	81.82%	76.28%
07	59,109	3,688	6.65%	60.54%	2.76	4.38%	82.23%	76.72%
08	58,981	3,698	6.90%	60.41%	2.74	3.79%	81.48%	75.82%
09	59,143	3,743	6.71%	60.69%	2.77	5.53%	81.56%	76.13%
Ave:	59,081	3,684	6.79%	60.43%	2.76	4.39%	81.84%	76.27%

Table 7.8: Statistics for the nine test corpora used for evaluation

the tenth test corpora was used for development of *IceTagger* and *IceMorphy*. The tagging accuracy for unknown words, known words and all words, for the tenth corpus, which we do not include in the average accuracy tagging figures, is 80.52%, 93.51% and 92.63%, respectively.

In an evaluation of tagging accuracy it is common to present figures for the somewhat unrealistic case of a closed vocabulary, i.e. assuming no existence of unknown words. When we evaluated *IceTagger* under this assumption the average accuracy was 93.84%.

As discussed in Section 7.2.1, named entity information is included in the tags for proper nouns in the Icelandic tagset. This is unusual and is normally not part of other tagsets, probably because this feature is not of syntactic nature. Table 7.10 shows the tagging accuracy of *IceTagger* when this feature is ignored. By comparing Tables 7.9 and 7.10, it can be seen that the overall tagging accuracy gain of 0.08% is mostly attributed to an increase in tagging accuracy of unknown words. This was to be expected, because guessing named entity information for unknown words is difficult, and indeed not a part of *IceMorphy* (a known proper noun is usually not

w assuming that t is not the only tag left for w. When tags are checked in the ascending order of frequency, a frequent tag, which might indeed be the correct tag, has a better chance of "surviving", because more infrequent tags are eliminated first.

Test corpus	unknown words	known words	all words	closed vocabulary
01	74.77%	92.72%	91.36%	93.82%
02	74.94%	92.57%	91.38%	93.76%
03	73.48%	92.81%	91.49%	93.80%
04	75.70%	92.73%	91.60%	93.83%
05	76.67%	92.98%	91.93%	94.08%
06	75.81%	92.79%	91.66%	93.94%
07	73.99%	92.88%	91.62%	93.95%
08	74.82%	92.63%	91.40%	93.77%
09	75.61%	92.51%	91.38%	93.64%
Ave:	75.09%	92.74%	91.54%	93.84%

Table 7.9: Tagging accuracy of *IceTagger* for the nine test corpora used for evaluation

ambiguous with respect to the named entity feature).

In Section 7.2.2, we stated that the heuristics of *IceTagger* have a large impact on the overall tagging accuracy. This can be seen by comparing Tables 7.9 (accuracy numbers obtained using both the local rules and the heuristics) and 7.11 (accuracy numbers obtained using only the local rules).

Recall that, for words that have still not yet been fully disambiguated, the default heuristic of *IceTagger* is simply to choose the most frequent tag. Table 7.12 shows the ambiguity rate, precision and recall for each test corpus when allowing *IceTagger* to generate ambiguous output, instead of simply selecting the most frequent tag when further disambiguation is not possible. Ambiguity rate is the average number of tags per word, and we define precision and recall in the following manner:

$$precision = \frac{\#\text{ of correct proposed tags}}{\#\text{ of proposed tags}} \tag{7.4}$$

$$recall = \frac{\#\text{ of correct proposed tags}}{\#\text{ of tokens}} \tag{7.5}$$

According to Table 7.12, the average recall is 93.77% compared to the

Test corpus	unknown words	known words	all words
01	75.67%	92.75%	91.46%
02	75.99%	92.59%	91.47%
03	74.13%	92.82%	91.55%
04	76.70%	92.75%	91.68%
05	77.77%	92.99%	92.01%
06	76.62%	92.81%	91.73%
07	75.13%	92.89%	91.71%
08	75.70%	92.65%	91.48%
09	76.57%	92.53%	91.45%
Ave:	76.03%	92.75%	91.62%

Table 7.10: Tagging accuracy of *IceTagger* for the nine test corpora when ignoring named entity information

average accuracy[5] of 91.54%, when applying the default heuristic (see Table 7.9). On the other hand, average precision drops dramatically, i.e. from 91.54% to only 72.62%.

The low precision can be explained by the relatively high average ambiguity rate of 1.29. Voutilainen (1995), for example, reports an ambiguity rate of 1.04-1.08 (and 99.7% recall) for the EngCG parser. The precision of *IceTagger* could be increased by writing more local rules, but achieving comparable figures to the EngCG parser would of course demand considerable effort.

Table 7.13 shows the average tagging accuracy of *IceTagger* for different word classes of unknown words. Highest accuracy is obtained for unknown common nouns, for which *IceMorphy* is indeed able to achieve highest recall (see Table 7.3).

We have previously stated that morphological analysis is more accurate than ending analysis. This fact is reflected in the accuracy of unknown words in *IceTagger* using different components of *IceMorphy*; see Table 7.14. On the average, the morphological analyser produces results for 40.22% of unknown words and the average tagging accuracy for morphologically analysed words is 84.74%. In contrast, the ending analyser produces analysis for 26.29% of

[5]Note that accuracy=recall=precision when full disambiguation is carried out.

Test corpus	unknown words	known words	all words
01	63.31%	86.68%	84.92%
02	64.87%	86.71%	85.24%
03	63.42%	87.04%	85.43%
04	66.11%	87.32%	85.91%
05	65.82%	87.35%	85.96%
06	65.91%	86.77%	85.38%
07	63.07%	87.44%	85.82%
08	64.47%	87.01%	85.45%
09	65.48%	86.66%	85.23%
Ave:	64.72%	87.00%	85.48%

Table 7.11: Tagging accuracy of *IceTagger* for the nine test corpora when not using the heuristics

		Precision			Recall		
Test corpus	Ambig. rate	unknown words	known words	all words	unknown words	known words	all words
01	1.29	50.24%	74.72%	72.42%	81.07%	94.72%	93.69%
02	1.29	50.86%	74.70%	72.72%	80.72%	94.60%	93.67%
03	1.29	47.91%	74.80%	72.47%	78.78%	94.72%	93.63%
04	1.29	51.25%	74.55%	72.66%	80.70%	94.66%	93.73%
05	1.28	51.39%	75.28%	73.35%	82.04%	94.88%	94.05%
06	1.29	51.99%	74.80%	72.96%	80.98%	94.75%	93.83%
07	1.30	48.72%	74.57%	72.38%	80.55%	94.82%	93.87%
08	1.29	51.17%	74.39%	72.43%	80.96%	94.69%	93.74%
09	1.30	53.80%	73.73%	72.18%	81.28%	94.60%	93.71%
Ave:	1.29	50.81%	74.62%	72.62%	80.79%	94.72%	93.77%

Table 7.12: Ambiguity rate, precision and recall of *IceTagger* for the nine test corpora used for evaluation when allowing ambiguous output

Common nouns	Proper nouns	Adjectives	Verbs	All words
81.13% (57.79%)	56.75% (10.87%)	75.73% (18.26%)	69.90% (9.03%)	75.09%

Table 7.13: Average tagging accuracy of *IceTagger* for different word classes of unknown words. The figures in parenthesis show how often each class occurs

Morphological analysis	Compound analysis	Ending analysis	Morphological accuracy	Compound accuracy	Ending accuracy
40.22%	31.10%	26.29%	84.74%	78.57%	57.09%

Table 7.14: Ratio of unknown words that are successfully analysed by components of *IceMorphy* and the corresponding tagging accuracy of *IceTagger* for these words

unknown words resulting in only 57.09% average tagging accuracy. Bear in mind, that the ending analyser is only applied if morphological/compound analysis fails, and therefore, to some extent, the ending analyser handles the more difficult cases.

Earlier, we explained the large difference in tagging accuracy when tagging Icelandic text vs. English text by the size of the Icelandic tagset used. Accordingly, one would expect the tagging accuracy of Icelandic to get closer to the accuracy published for English when condensing the Icelandic tagset. We tested this by ignoring the case and gender features as well as the named entity feature of proper nouns in the tags when calculating the accuracy, effectively resulting in a set of 99 tags. Indeed, Table 7.15 shows that when using a much smaller tagset, the tagging accuracy falls in the 95-97% range as reported for English. Note, however, that the case feature plays an important role in Icelandic and, therefore, valuable information is lost if this feature is ignored.

Words	Accuracy
Unknown words	86.51%
Known words	96.25%
All words	95.59%
Closed vocabulary assumption	96.86%

Table 7.15: Average tagging accuracy of *IceTagger* using a condensed tagset of 99 tags

7.2.3.1 Distribution of tagging errors

In what follows, we use the following definitions: A *tag type error* $A>B$ occurs when tag A is proposed by the tagger, but tag B is the correct tag. A word error occurs when a word is incorrectly tagged by the combined tagger.

We have previously pointed out (in Section 5.3.1) that the most frequent ambiguous word forms are responsible for a large amount of the total ambiguity. Table 7.16 shows the distribution of the most frequent errors made by *IceTagger* for all of the 9 test corpora. The table shows both proposed tags by *IceTagger* vs. correct tags (i.e. a tag type error), as well as individual words responsible for most errors. The table shows that the 25 most frequent tag type errors (of 5010 different tag errors made) are responsible for 21.87% of the total errors, and the 25 words that are responsible for 22.62% of the total errors. It is notable that the 158 most frequent tag type errors account for 50% of the total errors and only 198 words are responsible for 40% of the total errors (not shown in table).

Six out of the seven most frequent tag type errors have to do with prepositions. Either *IceTagger* proposes the wrong case for a preposition (e.g. aþ>ao), proposes an adverb instead of a preposition (e.g. aa>ao) or proposes a preposition instead of an adverb (ao>aa). Additionally, a part of the fourth most frequent tag error (nveþ>nveo) can be attributed to a previous wrongly selected preposition.

Not surprisingly, the words responsible for the most frequent tag errors are function words like prepositions and conjunctions ("*á*", "*í*", "*að*", "*sem*", "*við*", "*með*"), but also pronouns belonging to more than one pronoun category, like "*það*", "*þeim*", "*því*", and "*þeirra*" (all of which can be either a personal or a demonstrative pronoun).

The information in Table 7.16 is valuable because it can be used to build

more relevant local or global rules. Nevertheless, in many cases, world knowledge or semantic knowledge is needed to correct these errors.

For example, consider the sentence "*ég hafði lofað að koma með hafurtask mitt á heimili hjónanna*" (*I had promised to bring my stuff to the couple's home*). In this context, the word "*á*" can be a preposition governing the accusative or the dative case. *IceTagger* tries to disambiguate prepositions by considering the cases of the following words. Unfortunately, the very common problem here is that the following neuter noun "*heimili*" (home) has identical forms in the accusative and dative cases. The correct preposition, in this instance, is the accusative. However, in the quite similar sentence "*ég hafði lofað að koma með hafurtask mitt á hesti hjónanna*" (*I had promised to bring my stuff on the couple's horse*) the correct case is dative because the stuff was brought *ON a horse* in contrast to bringing the stuff *TO the couple's home* in the former sentence.

7.2.4 Comparison with data-driven taggers

In this section, we compare the results achieved by *IceTagger* with the results obtained by the DDTs described in chapter 2.3.4: *fnTBL* (Ngai and Florian 2001), *MXPOST* (Ratnaparkhi 1996), *MBT* (Daelemans et al. 1996), *TnT* (Brants 2000) and *TriTagger* (our re-implementation of *TnT*). Let us, first, briefly recollect the description of the individual taggers.

The *fnTBL* tagger is a fast implementation (in C and Perl) of transformation-based error-driven learning (TBL). In TBL, the training phase consists of, first, assigning each word its most likely tag without regard to context, and, second, learning a set of ordered rules which transform a tag X to a tag Y, with regard to context. New text is then tagged by applying the rules in the correct order.

The *MXPOST* tagger (implemented in Java) uses a binary feature representation to model tagging decisions, where each feature encodes any information that can be used to predict the tag for a particular word. The goal of the model is to maximise the entropy of a distribution, subject to certain feature constraints.

A memory-based model is used in the *MBT* tagger (implemented in C++). During training, a feature representation of an instance (word and its context) along with its correct tag (target class) is simply stored in memory. New instances are then tagged by retrieving the tag from the most similar instances in memory.

Proposed tag > correct tag	Error rate	Cumulative rate	Word error	English translation[a]	Error rate	Cumulative rate
aþ>ao	2.58%	2.58%	á	on	2.51%	2.51%
ao>aþ	1.62%	4.21%	í	in	2.29%	4.80%
aa>ao	1.54%	5.74%	að	to	1.73%	6.52%
nveþ>nveo	1.41%	7.15%	sem	that	1.44%	7.97%
aa>aþ	1.32%	8.47%	það	it	1.32%	9.29%
ao>aa	0.96%	9.43%	við	at	1.16%	10.45%
sfg3fn>sng	0.94%	10.37%	sér	himself	1.14%	11.59%
ct>c	0.93%	11.29%	þeim	they	1.10%	12.68%
nveo>nveþ	0.84%	12.13%	því	it	0.89%	13.57%
nhen>nheo	0.79%	12.92%	með	with	0.83%	14.40%
nkeþ>nkeo	0.76%	13.67%	þá	then	0.80%	15.20%
nheo>nhen	0.69%	14.36%	hvað	what	0.74%	15.93%
c>aa	0.67%	15.04%	þeirra	theirs	0.69%	16.63%
aa>lhensf	0.66%	15.70%	til	to	0.67%	17.29%
sng>sfg3fn	0.64%	16.34%	fyrir	for	0.65%	17.95%
lhensf>aa	0.61%	16.95%	eftir	after	0.62%	18.57%
nkeo>nkeþ	0.61%	17.55%	er	is	0.61%	19.18%
aþ>c	0.61%	18.16%	um	about	0.58%	19.76%
foheo>fohen	0.59%	18.75%	hann	he	0.50%	20.26%
aþ>aa	0.54%	19.29%	sig	himself	0.47%	20.73%
ssg>sþghen	0.54%	19.83%	einn	one	0.43%	21.16%
fohen>foheo	0.54%	20.36%	mikið	much	0.42%	21.57%
sfg3eþ>svg3eþ	0.51%	20.88%	eitt	one	0.38%	21.95%
nvfn>nvfo	0.50%	21.38%	heldur	but	0.35%	22.30%
fphfþ>fpkfþ	0.49%	21.87%	einu	one	0.31%	22.62%

Table 7.16: The 25 most frequent errors made by *IceTagger*

[a] Corresponding to the most frequent tag.

The *TnT* tagger (a very fast C implementation) uses a second order (trigram) probabilistic Hidden Markov Model (HMM). The probabilities of the model are estimated from a training corpus using maximum likelihood estimation. New assignments of POS to words is found by optimising the product of lexical probabilities ($p(w_i|t_j)$) and contextual probabilities ($p(t_i|t_{i-1}, t_{i-2})$) (where w_i and t_i are word i and tag i, respectively).

TriTagger is our re-implementation (in Java and Perl) of the functionality of the *TnT* tagger. The difference between these two taggers is that *Tri* uses the same list of idioms as *IceTagger* and the special dictionary described in Section 7.2.3, as a backup dictionary. We use this tagger for integration with *IceTagger* (see Section 7.3).

All the DDTs use some kind of a suffix analysis to guess the tag profile for unknown words. *IceTagger*, however, uses the integrated morphological analyser, *IceMorphy*, to obtain the tag profile for unknown words.

The DDTs were trained and tested (with their default options) on the *IFD* corpus. We used the same training and test corpora pairs, compiled for use in ten-fold cross-validation, as described in Section 4.1.1. In order to facilitate fair comparison with *IceTagger*, we only present accuracy figures computed using the first nine test corpora (recall that the tenth test corpus was used for development of *IceTagger*).

In what follows (especially in the tables) we use *MXP* for *MXPOST*, *TBL* for *fnTBL*, *Tri* for *TriTagger*, and *Ice* for *IceTagger*.

The tagging results are shown in Table 7.17 — here, we summarise the main results:

- *IceTagger* achieves highest average accuracy figures for unknown words, known words and all words. Three out of every four unknown words receive the correct analysis by *IceTagger*.

- The average overall tagging accuracy of *IceTagger* is 91.54% compared to 90.44% for the TnT tagger. Overall, according to our results, *IceTagger* makes 11.5% less errors than the TnT tagger.

- The accuracy of the *TnT* (or Tri) tagger for unknown words is remarkably good. The *TnT* tagger is a language independent tagger and has, thus, no knowledge of Icelandic morphology. *TnT* bases its assignment of tags to unknown words solely on suffix distribution (i.e. lexical probabilities) derived from a training corpus, and how well a given tag "fits in" with neighbouring tags (i.e. contextual probabilities).

Words/Tagger	Base[a]	MXP	MBT	TBL	TnT	Tri	Ice
Unknown	4.39%	62.29%	59.40%	55.51%	71.68%	71.04%	75.09%
Known	81.84%	91.00%	91.47%	91.82%	91.82%	91.87%	92.74%
All[b]	76.27%	89.03%	89.28%	89.33%	90.44%	90.46%	**91.54%**
Δ_{Err}[c]		53.77%	54.83%	55.04%	59.71%	59.80%	64.35%
Training time[d]		16,200	350	8,100	3	23	
Testing time		260	1,590	135	3	71	30

Table 7.17: The average tagging accuracy of Icelandic text using various taggers

[a] A baseline tagger which assigns each known word its most frequent tag, and the most frequent noun tag/proper noun tag to lower case/upper case unknown words.

[b] Our results for the MXP, TBL and TnT taggers are slightly different from the results in (Helgadóttir 2004), which can probably be explained by different parameter settings.

[c] Error reduction with regard to the errors made by the baseline tagger.

[d] Training and testing time is measured in seconds (on an Intel Pentium 1.6Ghz, with 512MB of RAM) for the first training (531k) and test corpora (59k) pair of the *IFD* corpus. *IceTagger* (*Ice*) does not need a training phase.

- The *fnTBL* obtains the same accuracy for known words as the *TnT* tagger, but is the least effective of all the taggers for unknown words.
- The *TnT* tagger is by far the fastest tagger, both for training and testing. To some extent, this is due to the implementation language, i.e. the tagger is written in C, whereas, for example, *IceTagger* is written in Java.
- For each test corpus, the difference between *IceTagger* and the closest competitor (TnT)[6] is statistically significant ($\alpha < 0.005$, using McNemar's chi-squared goodness-of-fit test as described by Dietterich (1998)).

7.2.5 Using other corpora

When evaluating tagging accuracy it is important, if at all possible, to test taggers on text types that the taggers have not been trained on (in the case

[6] Since the *Tri* tagger is a re-implementation of the *TnT* tagger, we consider the latter as the main competitor.

Test corpus	# of tokens	# of sentences	unknown word ratio
young	3,881	234	7.21%
old	6,023	226	8.88%
law/business	2,778	134	13.97%
computers	2,926	142	15.07%
newspaper	10,016	500	11.08%

Table 7.18: Statistics for the other corpora

	young literature			*old* literature		
Tagger	Unknown words	Known words	All words	Unknown words	Known words	All words
Ice	74.64%	93.03%	91.70%	72.90%	92.78%	91.02%
TnT	76.43%	92.25%	91.11%	72.86%	93.36%	91.53%
TBL	57.50%	91.17%	88.74%	59.11%	91.92%	88.99%
MBT	60.36%	90.86%	88.66%	57.81%	91.23%	88.25%
MXP	65.00%	90.39%	88.56%	64.31%	90.43%	88.10%

Table 7.19: Tagging accuracy for the *young* and *old* corpora

of DDT taggers), or on text types that have not been used for development of the taggers (in the case of linguistic rule-based taggers).

In this section, we present evaluation results for all the previously used taggers, using the text segments described in Section 4.1.2. The segments consist of 5 types: *young* literary work, *old* literary work, *law/business* text, text about *computers* and *newspaper* text. For convenience, the statistics on the other corpora from Section 4.1.2 is reproduced in Table 7.18.

We trained each DDT using the whole *IFD* corpus as the training corpus and tested each tagger on each of the text segments. Note that since *IceTagger* is not a DDT it does not need a training phase. The results are summarised in tables 7.19 — 7.23.

From tables 7.19 – 7.23, we can deduce the following:

- *IceTagger* achieves the highest accuracy for four out of the five different text segments — it is only beaten by the *TnT* tagger when tagging the *old* corpus. The reason for lower tagging accuracy of *IceTagger*

	law/business texts			*computer* texts		
Tagger	Unknown words	Known words	All words	Unknown words	Known words	All words
Ice	75.77%	90.50%	88.44%	51.47%	91.31%	85.30%
TnT	70.69%	88.91%	86.36%	48.98%	90.91%	84.59%
TBL	53.47%	87.15%	82.43%	37.64%	89.13%	81.37%
MBT	58.10%	87.44%	83.33%	41.72%	88.89%	81.78%
MXP	60.93%	87.15%	83.48%	44.22%	88.37%	81.72%

Table 7.20: Tagging accuracy for the business/law texts and computer texts

	newspaper texts		
Tagger	Unknown words	Known words	All words
Ice	71.71%	93.45%	91.04%
TnT	64.75%	89.44%	86.70%
TBL	50.45%	90.39%	85.95%
MBT	53.15%	89.16%	85.16%
MXP	55.04%	88.86%	85.10%

Table 7.21: Tagging accuracy for the newspaper texts

Test corpus	Common nouns	Proper nouns	Adjectives	Verbs	All words
young	79.63% (57.86%)	77.27% (7.86%)	70.21% (16.79%)	73.17% (14.64%)	74.64%
old	78.04% (55.33%)	82.98% (8.79%)	73.03% (16.64%)	64.10% (14.58%)	72.90%
law/business	80.80% (71.13%)	54.55% (11.34%)	86.87% (7.73%)	55.56% (4.64%)	75.77%
computers	79.08% (44.44%)	22.76% (32.88%)	51.43% (7.94%)	52.94% (3.85%)	51.47%
newspaper	82.94% (53.87%)	44.83% (23.51%)	80.92% (11.80%)	63.53% (7.66%)	71.71%

Table 7.22: Tagging accuracy of *IceTagger* for different word classes of unknown words. The figures in parenthesis show how often each class occurs

Test corpus	Morpho analysis	Compound analysis	Ending analysis	Morpho accuracy	Compound accuracy	Ending accuracy
young	34.29%	31.07%	31.43%	80.21%	80.46%	69.32%
old	34.02%	32.90%	31.59%	78.02%	76.70%	65.68%
law/business	34.02%	45.36%	16.24%	82.58%	77.84%	58.73%
computers	17.46%	31.75%	36.73%	80.52%	70.00%	33.33%
newspaper	28.92%	33.33%	30.45%	84.74%	81.62%	57.40%

Table 7.23: The table shows the ratio of unknown words that are successfully analysed by components of *IceMorphy*, and the corresponding tagging accuracy of *IceTagger* for these words

compared to *TnT* for this corpus is the surprisingly high accuracy of the latter tagger for known words (93.36%). This accuracy is much higher than the corresponding accuracy (91.82%) of the *TnT* tagger when tagging the *IFD* corpus (see Table 7.17). This is difficult to explain, but bear in mind that, in this experiment, the size of the test corpus is small compared to the test corpora sizes we used when testing against the *IFD* corpus.

- The data-driven taggers achieve comparable tagging accuracy on the newspaper text (see Table 7.21). *IceTagger*, however, obtains much higher accuracy than the other taggers (error reduction with regard to the *TnT* tagger is 32.6%). The most probable explanation for this is that the DDTs have been trained on material which is different from the testing material, i.e. the *IFD* corpus does not include newspaper texts. The results thus indicate that the performance of the DDTs is sensitive to its training material. *IceTagger* was, on the other hand, developed by building linguistically motivated rules, which should be better applicable to any type of text.

- The tagging accuracy of *IceTagger* decreases as the ratio of unknown words in the test corpora increases. For the *young*, *old*, *computers*, *law/business* and *computers* corpora, the unknown word ratios are 7.21%, 8.88%, 11.08%, 13.97% and 15.07%, respectively, and the tagging accuracy for these corpora is 91.70%, 91.02%, 91.04%, 88.44% and 85.30%, respectively.

- For all the taggers, the tagging accuracy of unknown words in the computer texts is substantially lower than for unknown words in the other texts. This can, first, be explained by the large ratio of unknown proper nouns (technical acronyms) in these texts (and the small ratio of common nouns), compared to the other texts (see Table 7.22). Secondly, the *IFD* corpus, which the DDTs were trained on, does not include any technical texts and the DDTs have, thus, not been able to learn from such texts. *IceTagger* has similar problems in this regard because its unknown word guesser, *IceMorphy*, was developed with regard to a part of the *IFD* corpus.

7.2.6 Conclusion

In this chapter, we have evaluated individual components of *IceTagger* and compared its overall tagging accuracy to corresponding accuracy obtained by DDTs.

Evaluation of *IceMorphy*, the unknown word guesser used in *IceTagger*, showed that its precision, when guessing possible tags for all unknown words, is just above 58%. The recall, however, whose accuracy is more important for a tagger, is close to 76%. Note, however, that with a more comprehensive dictionary, both the precision and the recall of *IceMorphy* should improve. We would like to verify this hypothesis in future work.

Our evaluation results for *IceTagger* shows that it obtains higher accuracy than five state-of-the art DDTs, when tested against the *IFD* corpus. Using this corpus, our tagger makes 11.5% less errors than the best performing DDT. Furthermore, for four out of five other different test corpora, *IceTagger* obtains the highest accuracy.

The disambiguator of *IceTagger* uses only about 175 local rules, but is able to achieve high accuracy through the use of global heuristics along with automatic tag profile gap filling. The heuristics guess the functional roles of the words in a sentence, mark prepositional phrases and use the acquired knowledge to force feature agreement where appropriate. Other morphologically complex languages might use similar heuristics for POS tagging.

Contrary to previous experience and assumptions, our work shows that a linguistic rule-based approach does not have to be very labour intensive in order to achieve high tagging accuracy. In the design of the disambiguation phase of *IceTagger*, main emphasis was put on developing the heuristics, instead of writing a large set of constraint rules. Moreover, the main dictionary used by *IceTagger* is automatically derived from the *IFD* corpus. This is the main reason why the development of the system took only 7 man months. In Section 5.3.1, we stated that the aim of developing *IceTagger* was to show that a linguistic rule-based tagger could be developed, without an enormous effort, which, due to data sparseness problems, could achieve higher accuracy than a state-of-the-art data-driven tagger. Our evaluation shows that we did indeed succeed.

We are convinced that the tagging accuracy of *IceTagger* can still be improved, by improving each of its individual components. For example, writing more local rules, to handle the frequent ambiguous word forms, will certainly be beneficial. The heuristics can also be improved, as discussed in

Section 7.2.2.1. Trying to improve the accuracy of a DDT, however, demands a significantly larger training corpus, which is time-consuming to construct.

The main dictionary used by *IceTagger* is derived from a training corpus. Using the whole *IFD* corpus, the dictionary consists of about 60,000 word forms. In future work, we would like to evaluate the tagger using a much larger dictionary. Such a dictionary can be constructed using data from *Beygingarlýsing íslensks nútímamáls* (*Morphological Description of Icelandic*; (Bjarnadóttir 2005)), a resource currently containing about 250,000 lemmas and 5.5 million word forms. Due to size, it is imperative that such a dictionary is represented using an appropriate data structure, like a trie, or converting it to a DFA. By using a larger dictionary, the ratio of unknown words should decrease and result in higher tagging accuracy. On the other hand, the average ambiguity rate could increase (on the average, more tags should be found for each word form), which itself should result in lower tagging accuracy.

The tagset used in this research is large, i.e. consists of about 660 tags. In future work, we would like to design a smaller version of this tagset (in fact, we experimented with a smaller tagset in Section 7.2.3). When designing the smaller tagset, the main decision is what features of the large tagset can be left out without to much loss of information. In fact, the large tagset does include information which are not of syntactic nature, i.e. the last character for proper nouns which encodes named entity information. Moreover, we will later see, in our work on shallow parsing (see chapter 6), that a substantial part of the features can be omitted for the purpose of shallow parsing.

7.3 Integration of taggers

In this section, we describe four integration methods, all of which have resulted in an improved tagging accuracy of Icelandic text.

The first three methods consist of integrating our morphological analyser with state-of-the-art DDTs. The last method consists of integrating our trigram tagger *Tri* with *IceTagger*.

As can been seen in Table 7.17, the tagging accuracy of *fnTBL* for unknown words is relatively poor (55.51%). In order to improve this accuracy, we overwrote *fnTBL's* default initial tagging assignment for unknown words (which assigns the most probable common noun tag (proper noun tag) to an unknown lower case word (upper case word)) by calling *IceMorphy* instead

(this was possible due to the availability of *fnTBL's* source files). We made *IceMorphy* return the most probable tag (according to the relevant training corpus) from the set of guessed tags for the given unknown word (recall that a transformation-based method needs one single tag for its initial assignment). This increased the overall accuracy of *fnTBL* from 89.33% to 90.15% (compare tables 7.17 and 7.24). The total accuracy obtained is though still lower than the corresponding accuracy by the *TnT* tagger.

It is noteworthy that the tagging accuracy of *IceTagger* for known words is substantially higher than the corresponding accuracy of the next best tagger, *TnT*, as is evident in Table 7.17. This can partly be explained by the following observation. The dictionary derived from a training corpus has, unfortunately, a number of tag profile gaps. That is, for a given word w_i, with tags t_{i1}, t_{i2}, ..., t_{ij}, the correct tag profile, in fact, could have more than j tags (the missing tags might just not have been encountered during training (derivation of the dictionary). *IceMorphy* is able to generate missing tags in a tag profile for a word belonging to a particular morphological class (see Section 5.2.5), and *IceTagger* benefits from this functionality of *IceMorphy*. In our experiments, we had noted that the tag profile gap filling component contributes about 0.85% to the total average accuracy of *IceTagger*.

Our second integration method consists of using this feature of *IceMorphy* to generate a "filled" dictionary, to be used by another tagger, in this case the *TnT* tagger. Each record in the dictionary used by *TnT* consists of a word and the corresponding tags found in the training corpus. Additionally, to facilitate lexical probability calculations, each tag is marked by its frequency (i.e. how often the tag appeared as a label for the given word). Using the same training corpus, we made *IceMorphy* generate a "filled" dictionary such that each generated missing tag was marked with the frequency 1[7].

Consequently, by making the *TnT* tagger use this enhanced dictionary, the tagging accuracy improved from 90.44% to 91.18% (compare tables 7.17 and 7.24).

The third integration method is an integration of our *Tri* tagger with *IceMorphy*. In order to improve the accuracy of this tagger, we call *IceMorphy* from within the *Tri* tagger to obtain possible tags for unknown words. We made the *Tri* tagger only use the tags for those unknown words that go

[7]This seems logical since the missing tags were not found in the training corpus and are, hence, infrequent. Admittedly, this is a very simple smoothing strategy and experimenting with more sophisticated smoothing strategies would be worthwhile.

successfully through morphological/compound analysis, but not those that go through ending analysis because, in *IceMorphy*, the former is much more accurate than the latter (see Table 7.4). Moreover, we made the *Tri* tagger benefit from the tag profile gap filling described above. This version of the *Tri* tagger achieves an accuracy of 91.34% (see Table 7.24).

The last integration method we have used is the integration of our linguistic rule-based tagger with the *Tri* tagger. By making *IceTagger* call the *Tri* tagger for full disambiguation (instead of simply selecting the most frequent tag for a word not fully disambiguated) the overall tagging accuracy increases from 91.54% to 91.80% (compare tables 7.17 and 7.24). The average ambiguity rate (tags per token) before calling the *Tri* tagger is 1.29.

Combining rule-based methods with statistical methods is well known. In one study, a HMM tagger and a linguistic rule-based tagger were used independently, their outputs aligned, and the result of the HMM tagger used to solve remaining ambiguities in the output of the rule-based tagger (Tapanainen and Voutilainen 1994). In another experiment, similar taggers were used one after the other, but the HMM tagger was trained on the output of the linguistic rule-based tagger (Ezeiza et al. 1998).

Note the difference between our integration method and the combination used in these two papers. Our integrated system runs like a single tagger, i.e. the text to be tagged is tagged only once. In contrast, the text is tagged twice in the combination methods, and, additionally, post-processed using alignment or training. A similar integration method as ours, using a linguistic rule-based tagger and a HMM tagger, has, in fact, been used for tagging Czech text (Hajič et al. 2001).

Henceforth, we will refer to the *fnTBL+IceMorphy* tagger as *TBL**, the *TnT+IceMorphy* tagger as *TnT**, the *Tri+IceMorphy tagger* as *Tri** and the *IceTagger+Tri* tagger as *Ice**.

7.3.1 Conclusion

We have described several tagger integration methods for the purpose of improving the tagging accuracy of Icelandic text. We have defined tagger integration as enabling one tagger to use a feature or a functionality of another tagger. The accuracy of the best performing integrated tagger, consisting of using *IceTagger*, for initial disambiguation, along with *Tri*, a tagger based on a HMM, for full disambiguation, is 91.80%.

Words/Tagger	TBL*	TnT*	Tri*	Ice*
Unknown words	66.30%	72.80%	74.46%	75.33%
Known words	91.90%	92.54%	92.58%	93.00%
All words	90.15%	91.18%	91.34%	**91.80%**
Δ_{Err}[a]	7.69%	7.74%	9.13%	3.07%

Table 7.24: Accuracy using integration of taggers

[a]Error reduction with regard to the errors made by the unchanged version of the corresponding tagger.

7.4 Combination of taggers

In the first tagger combination experiment for Icelandic, the *MXP*, *TBL* and *TnT* taggers were used in a simple voting scheme (see Section 7.1.1), obtaining an average accuracy of 91.54% (using all ten test corpora of the *IFD* corpus) (Helgadóttir 2004) (see row 1 of Table 7.25). In this section, we will combine unchanged version of taggers and integrated taggers to improve upon this previously published result.

7.4.1 Simple voting

Our first combined tagger consists of using *TBL*, *TnT* and *Ice* in a simple voting scheme (in the case where all the three taggers disagree the tag proposed by *Ice* is selected). We, thus, substituted the relatively low accuracy tagger *MXP* for *Ice*. This substantially improves the tagging accuracy, from 91.54% to 92.61% (see row 2 of Table 7.25). By adding the two least accurate taggers, *MXP* and *MBT*, to the combination pool, the overall accuracy increases further to 92.80% (see row 3). This supports the hypothesis that adding different taggers to a combination pool generally increases the accuracy. In the case of (2:2:1) ties, we select the tag belonging to the most accurate tagger in the tie, and when all taggers disagree we select the tag proposed by the most accurate tagger, *Ice*.

Next, we used simple voting to combine *TBL**, *TnT** and *Ice* (recall that the first two are improved versions of *TBL* and *TnT* (see Section 7.3)). Not surprisingly, this combination improves the tagging accuracy from 92.61% to 92.94% (compare rows 2 and 4 in Table 7.25).

The accuracy of the combined tagger in row 4 was next improved by adding the taggers *MXP* and *MBT* to the combination pool, resulting in an accuracy increase to 93.29% (see row 5). This time, the addition of the two taggers is about twice as effective than before, mainly because of higher accuracy for unknown words. The errors made by these two taggers for unknown words are probably, in many cases, complementary to the corresponding errors proposed by *TBL** (which receives "help" from *IceMorphy* for unknown words), but less complementary to *TBL*, which was used in the combination pool in row 3.

The benefit of using our integrated taggers is clear by comparing the accuracy of the combined taggers in rows 2 and 4, and in rows 3 and 5, in Table 7.25.

Next, we replaced *Ice* with *Ice**, i.e. *Ice* with the *Tri* tagger for full disambiguation (recall that the accuracy of *Ice* is 91.54%, but accuracy of *Ice** is 91.80%). This slightly improved the overall tagging accuracy (see row 6 in Table 7.25).

Note that we have not used *Tri**, the second most accurate tagger, in our combination. We did not expect an improvement by adding *Tri** to the pool because of reason of similarity, i.e. *Tri** is a HMM tagger (as *TnT*) and uses *IceMorpy* for guessing possible tags for unknown words (as *IceTagger*). In fact, when we added *Tri** as the sixth tagger to the MXP+MBT+TBL*+TnT*+Ice* pool, the accuracy decreased from 93.34% to 93.21%. This result agrees with other research, e.g. (Sjöbergh 2003).

The first row of Table 7.26 shows the accuracy of the simple voting for tags selected by different number of votes. The figures in parenthesis show how often, on the average, each scenario occurs. For example, the scenario, in which three taggers select the same tag, occurs 7.14% of the time and the accuracy of the selection is 63.42%. As expected, the accuracy decreases with fewer votes behind the selected tag.

7.4.2 Weighting

Up to this point, we have only used simple voting when combining the taggers. The next logical step is to try more sophisticated voting schemes, e.g. weighted voting, in which we follow the work of van Halteren et al. (2001).

For weights, we, first, used the overall accuracy of each tagger (*TotPrecision*). This resulted, effectively, in no change in comparison with the best simple voting scheme; see row 7 in Table 7.25.

			Accuracy of words			
No.	Combination	Method[a]	Unkn.	Known	All[b]	Δ_{Err}[c]
1.	MXP+TBL+TnT	Simple	71.80%	92.99%	**91.54%**	12.2%
2.	TBL+TnT+Ice	Simple	76.76%	93.77%	**92.61%**	12.7%
3.	MXP+MBT+TBL+TnT+Ice	Simple	76.74%	93.97%	**92.80%**	14.9%
4.	TBL*+TnT*+Ice	Simple	76.55%	94.13%	**92.94%**	16.6%
5.	MXP+MBT+TBL*+TnT*+Ice	Simple	78.70%	94.36%	**93.29%**	20.7%
6.	MXP+MBT+TBL*+TnT*+Ice*	Simple	78.65%	94.41%	**93.34%**	18.8%
7.	MXP+MBT+TBL*+TnT*+Ice*	TotPrecision	78.56%	94.39%	93.32%	18.5%
8.	MXP+MBT+TBL*+TnT*+Ice*	TagPrecision	77.89%	94.28%	93.16%	16.6%
9.	MXP+MBT+TBL*+TnT*+Ice*	Precision–Recall	78.85%	94.39%	93.33%	18.7%
10.	MXP+MBT+TBL*+TnT*+Ice*	Simple+rule 1	78.65%	94.50%	**93.43%**	19.9%
11.	MXP+MBT+TBL*+TnT*+Ice*	Simple+rules 1&2	78.66%	94.56%	**93.48%**	20.5%

Table 7.25: Accuracy using combination of taggers

[a] *Simple* is majority voting, in which ties are resolved by selecting the tag of the most accurate tagger in the tie. *TotPrecision*, *TagPrecision*, *Precision–Recall* are weighting schemes, and *rule* refers to linguistically motivated rules.

[b] All improvements in tagging accuracy are significant at $\alpha < 0.05$, using McNemar's chi-squared goodness-of-fit test as described by (Dietterich 1998).

[c] Error reduction with regard to the errors made by the best single tagger in the corresponding combination pool.

Second, we tried weighing with the precision of each tagger for each tag (*TagPrecision*). Using these weights, the accuracy dropped to 93.16% (see row 8).

Last, we tried using weights constructed by forcing each tagger not only to use its precision for each tag, but, additionally, to add to the vote for tags suggested by the opposition, by the amount (1-recall) on the opposing tag (marked as *Precision–Recall* in row 9). The amount (1-recall) for a given tag signifies how often a tagger fails to recognise the tag. This weight mechanism did neither produce any improvements (see row 9)[8].

These results, when using weighted voting, are similar to other published results, e.g. using *TotPrecision* (Sjöbergh 2003), and using *TagPrecision* and *Precision-Recall* for a detailed tagset (van Halteren et al. 2001).

7.4.3 Linguistically motivated rules

The first row of Table 7.26 shows that the tagging accuracy of the combined tagger is high when all the five individual taggers agree on the vote (98.86%).

[8] The precision and recall figures used were computed using the development corpus.

		Number of votes[a]					Rules	
No.[b]	Method	5	4	3	2	1	Rule 1	Rule 2
6.	Simple	98.86% (79.78%)	84.46% (10.35%)	63.42% (7.14%)	45.63% (2.56%)	29.51% (0.17%)		
10.	Simple+rule 1	98.87% (79.61%)	85.34% (10.14%)	63.86% (6.98%)	45.57% (2.52%)	28.44% (0.17%)	70.90% (0.56%)	
11.	Simple+rules 1&2	98.87% (79.61%)	83.91% (8.70%)	61.73% (6.03%)	41.53% (2.20%)	25.81% (0.16%)	70.90% (0.56%)	87.36% (2.74%)

Table 7.26: Accuracy for different number of votes, and for the linguistic rules

[a] The figures in parenthesis show how often, on the average, each scenario occurs.
[b] The number refers to the corresponding row in table 7.25.

However, the accuracy falls rapidly with fewer and fewer votes behind the selected tag. When the selected tag receives four votes the accuracy has already dropped down to 84.46%, and even when there is a majority (3 votes) behind the selected tag, the accuracy is only 63.42%.

Linguistically motivated rules can help to improve the tagging accuracy of a combination method. We wrote two kinds of rules, both of which are based on specific strengths of *Ice*, and which are only fired if not all taggers agree upon a vote.

First, we have noticed that the DDTs have difficulties providing the correct tag for some specific tags in a particular context, whereas *Ice* performs considerably better for those tags in the same context. This occurs, for example, where there are "long" dependencies between a subject and a verb and the verb has the same lexical form for first and third person. A typical example (taken from the development corpus) is "*ég opnaði dyrnar, steig inn ...*" (I opened door, stepped inside ...). The verb "*steig*" should have the tag *sfg1eþ* (verb, indicative, active, first person, singular, past tense), but all the DDTs propose a third person verb (*sfg3eþ*). The reason is that the third person verb is more frequent and the DDTs have a limited context window size.

Another example of a long dependency is between a subject and a reflexive pronoun, e.g. "*... sagði konan og færði sig ...*" (...said woman and moved herself ...), in which the reflexive pronoun has the same lexical form in all genders. The correct tag for "*sig*" in this example is *fpveo*, but the DDTs provide the tag *fpkeo* which is more frequent.

In both these examples, *IceTagger* provides the correct tag because of its

built-in feature agreement functionality, but is outvoted by the DDTs. We, thus, built a simple rule which always selects the first person tags, if they are suggested by *Ice*, and the tags suggested by *Ice* for the reflexive pronouns "*sig*", "*sér*" and "*sín*". Despite the accuracy of this rule being only 70.90%, (see *Rule 1* in Table 7.26), the rule does improve the overall tagging accuracy because of the relatively low recall of the other taggers for these tags. For example, the average recall of the DDTs for the *sfg1eþ* tag (computed using the development corpus) is only 78.1%, compared to *Ice's* recall of 91.6% for the same tag.

For the second rule, we used a feature agreement constraint: "If all the tags, provided by the individual taggers for the current word, are nominal tags and the current tag provided by *Ice* agrees in gender, number and case with the preceding nominal tag (agreed upon by all taggers) or the following nominal tag (which has yet to be decided), then choose *Ice's* tag" (this is a slightly simplified version of our rule because we exclude specific nominal pairs). Using this rule improves the tagging accuracy because disambiguating using nominal feature agreement is one of the strengths of *Ice*. Table 7.26 shows that the accuracy of this rule (*Rule 2*) is 87.36%.

Row 11 of Table 7.25 shows that using simple voting along with the two linguistically motivated rules results in an tagging accuracy of 93.48%.

7.4.4 Discussion

7.4.4.1 Most frequent errors

For the test corpora, we computed the average maximum obtainable accuracy, 97.70%, assuming a voting method always selects the correct tag. Equivalently, this means that no tagger provides the correct tag for 2.30% of the tags. This number can be regarded as a measurement of the relative difficulty of the Icelandic tagging task. For Swedish, for example, no tagger is correct for only 1.2% of the tokens (using seven taggers and a tagset of size 150) (Sjöbergh 2003), and for English the corresponding figure is 0.78% (using four taggers and tagset size 170) (van Halteren et al. 2001).

Since there is a considerable gap between the highest accuracy using our combined tagger and the maximum obtainable accuracy, there should still be room for an improvement. In what follows, we use the definitions from Section 7.2.3.1 (here repeated for convenience): A tag type error $A>B$ occurs when tag A is proposed by the combined tagger, but tag B is the correct

tag. A word error occurs when a word is incorrectly tagged by the combined tagger.

When analysing the errors made by the best combined tagger, we noticed the following:

1. The 26 most frequent tag type errors are responsible for 25% of the total errors.

2. The four most frequent tag type errors consist of selecting the accusative case instead of dative or vice verse (7.8% of the total errors). These are the preposition tag type errors *aþ>ao* and *ao>aþ* and the noun errors *nveo>nveþ* and *nveþ>nveo*. Note that these errors occur very often in the same prepositional phrase, i.e. the combined tagger selects the wrong case for the preposition+noun pair.

3. The next two most frequent tag type errors are *aa>ao* and *ao>aa*, contributing 2.4% to the total errors. These are conflicts between adverbs (tag *aa*) and prepositions.

4. The 42 most frequent word errors constitute 25% of the total errors.

5. The two most frequent word errors are the prepositions "*í*" (in) and "*á*" (on) (4.5% of the total errors), both of which govern either the accusative case or the dative case (see item nr. 1 above). Moreover, the word "*á*" is very ambiguous, i.e. it can have one of eight tags *aþ_ ao_ sfg1en_ sfg3en_ aa_ nven_ nveo_ nveþ*.

6. The third most frequent word error is "*að*" (to) which can have one of four tags *cn_ c_ aþ_ aa*, denoting the infinitive marker, a conjunction, a preposition governing accusative and an adverb, respectively.

The above distribution of errors in the best combined tagger is very similar to the distribution of errors made by *Ice* (see Section 7.2.3.1).

One can deduce that in order to improve the tagging accuracy of Icelandic text the individual taggers need to perform better on the most frequent tag type and word errors. The problem is, however, that in many cases world knowledge or semantic knowledge is needed to fix these errors. This especially applies to the preposition tag type errors, in which a dative case is selected instead of an accusative case, and vice verse.

For the DDTs the only option is to add more training material, and for *IceTagger* more relevant linguistic rules need to be written.

7.4.4.2 Corpus annotation errors

We have previously pointed out that a combined tagger is useful for detecting annotation errors in a corpus. We manually examined the first 1000 errors made by our best performing simple voting method, when evaluated against the tenth test corpora (which, if you recall, has been used for development of *Ice*). Our examination revealed that 31 of the 1000 errors (i.e. 3.1%) were, in fact, annotation errors.

The first 1000 errors appeared in the first 18,398 tokens of the test corpus, i.e. 0.17% (31/18,398) of these tokens have been incorrectly annotated. If we assume that our sample is representative for the whole corpus, then this ratio is considerably higher than the corresponding ratio, 0.10%, computed using the English *LOB* corpus and manual examination of (only) 200 tokens (van Halteren et al. 2001). However, bear in mind, that the *IFD* tagset is almost 4 times larger than the *LOB* tagset.

7.4.5 Conclusion

We have described several combination methods for the purpose of improving tagging accuracy of Icelandic text. We combined five taggers, each based on a different language model. The best performing voting method (93.48%) consists of simple voting, in which ties are resolved in favour of better performing taggers, in addition to two simple linguistically motivated rules. This combination reduces the error rate by 20.5%, with regard to the best performing single tagger in the combination pool.

Error analysis showed that a small number of tag type errors is responsible for a large ratio of the total errors. Furthermore, we demonstrated that a small number of frequent words are responsible for these frequent tag type errors. Finally, we showed how a combination tagger can be used to estimate the annotation errors in a corpus.

Since the average obtainable maximum, computed using the test corpora, is 97.70%, we believe there is still room for improvement. There are several possibilities.

First, increasing the training corpus size. This is time-consuming, but might be a feasible option because our best combination method could be used for initial tagging, followed by manual corrections.

Second, adding more taggers to the combination pool might improve the tagging accuracy, assuming the taggers are different from those already in

our pool.

Third, adding more linguistic knowledge to *IceTagger* is possible, especially with the purpose of fixing frequent errors.

Fourth, the ratio of unknown words could be reduced by using an extensive dictionary, as described in Section 7.2.6.

Lastly, in our experiments, we have only used voting methods for combining taggers, but it has been shown that using stacking methods can improve the tagging accuracy further. In future work we would like to experiment with such methods, e.g. methods based on memory-based learning or maximum entropy.

Chapter 8

Parsing Icelandic Text

8.1 Introduction

We decided to use the *Parseval* scheme (see Section 2.4.6) as the evaluation method for *IceParser*. This scheme is the most common evaluation method, and even though no previous parsing results have been published for the Icelandic language (see Section 2.4.7) the widespread usage of this scheme makes it at least possible to compare our results to results published for related languages. Additionally, since our parser annotates constituent structure (and syntactic functions, each of which can be treated as a separate constituent during evaluation), it is straight-forward to use the *Parseval* scheme.

Since no Icelandic treebank exists, we needed to build a *gold standard* to be used in the evaluation. The *gold standard* consists of 509 sentences (8281 tokens), randomly selected from the *IFD* corpus. The *IFD* corpus is only POS tagged, and thus we manually annotated the sentences with constituent structure and syntactic functions, according to our annotation scheme (see Section 6.2). The manual annotation was carried out by two persons[1].

First, constituent structure was annotated, resulting in treebank A. Then, the output of *IceParser*, for constituent structure only, was evaluated against A. Second, syntactic functions were added to A and constituent structure removed, resulting in treebank B. Correspondingly, the output of *IceParser*, including syntactic functions, but with constituent structure removed, was

[1]Thanks to Einar Freyr Sigurðsson, a Masters student in linguistics at the University of Iceland, for helping the author with the annotation.

then evaluated against *B*.

8.2 Evaluation

We used the *Evalb* bracket scoring program (Sekine and Collins 1997) for automatic evaluation. For the evaluation of labelled constituent structure, we carried out two experiments. In the first one, we used the POS tags from the *IFD* corpus, i.e. we assumed correct tagging (see columns 2-4 in table 8.1). In the second one, we used the tags generated by *IceTagger* (see columns 5-7 in table 8.1).

In the case of correct tags, the overall F-measure (2*precision*recall / (precision+recall)) is 96.7%. As can be deduced from table 8.1, *VPx* (which stands for all subtypes of verb phrases; see Section 6.2.1.9), *CP*, *SCP* and *InjP* are "easy" to annotate. These phrase types constitute 28.6% of the phrases in treebank *A*, and thus help to make the overall accuracy quite high. On the other hand, the accuracy for the more "difficult" phrase types, like *AP*, *NP* and *PP* (which constitute 58.7% of the phrases), is about 95%-97%, according to our results.

The F-measure for only three types of phrases, *AdvP*, *APs* and *NPs*, is below 95%. At first sight, it might be surprising that the F-measure for adverb phrases is not higher. This can, however, be explained (and will be demonstrated in Section 8.3) by the fact that *IceParser* has problems deciding when two (or more) adjacent adverbs form a single phrase. The *APs* and *NPs* phrase types are used to group together a sequence of adjective phrases and noun phrases, respectively. In Section 8.3, we will show that annotating these phrase types is a relatively difficult task.

In the second experiment, we used *IceTagger* to tag the sentences in treebank *A*, before *IceParser* was run. The POS tagging accuracy for these sentences is 91.1% (unknown word ratio is 7.8%). In this case, the overall F-measure for constituent structure drops from 96.7% to 91.9% (see column 7 in table 8.1), which is equivalent to about 5.0% reduction in accuracy. The POS tagging accuracy is relatively low, compared to related languages (see discussion in Section 7.1.1.1), and this has substantial effect on the overall parsing accuracy.

As mentioned previously, we can not compare our results to other parsers for Icelandic, since this evaluation is the first parser evaluation published for

	Using correct POS tags			Using *IceTagger*			
Phrase type	Precision	Recall	F-measure	Precision	Recall	F-measure	Freq. in test data
AdvP	94.5%	89.2%	91.8%	86.3%	84.0%	85.1%	8.2%
AP	95.1%	95.2%	95.1%	85.4%	87.2%	86.3%	8.1%
APs	83.3%	90.9%	87.0%	64.9%	72.7%	68.6%	0.5%
NP	96.9%	96.7%	96.8%	93.2%	92.9%	93.0%	37.6%
NPs	78.9%	82.0%	80.4%	73.5%	75.0%	74.3%	1.5%
PP	96.7%	96.7%	96.7%	91.5%	91.1%	91.3%	13.0%
VPx	99.2%	99.3%	99.2%	93.7%	94.0%	93.8%	19.3%
CP	100.0%	100.0%	100.0%	99.7%	99.5%	99.6%	5.7%
SCP	100.0%	99.1%	99.6%	97.8%	97.4%	97.6%	3.4%
InjP	100.0%	100.0%	100.0%	100.0%	92.9%	96.3%	0.2%
MWE	93.9%	100.0%	96.9%	92.4%	92.9%	92.6%	2.5%
All	96.8%	96.6%	96.7%	92.0%	91.8%	91.9%	100.0%

Table 8.1: Accuracy for the various phrase types

the language. For the sake of a comparison with a related language[2], Swedish, Knutsson et al. (2003) report 88.7% F-measure for all phrases, and 91.4% for noun phrases, when using a tagger to preprocess the text and a shallow (not finite-state) rule-based parser. Using a finite-state parser, Kokkinakis and Johansson-Kokkinakis (1999) report higher numbers, 93.3% for all phrases and 96.2% for noun phrases, despite using a tagger for preprocessing. The tagger used, however, obtains very high accuracy when tagging the test data, i.e. 98.7%. We believe that this comparison indicates our parser performs well when annotating constituents.

For the evaluation of syntactic functions, we also carried out experiments with and without correct tagging[3]. When using the correct tags from the *IFD* corpus, the overall F-measure is 84.3% — see column 4 of table 8.2. When considering subjects and objects the highest accuracy is obtained for the functions *SUBJ*> and *OBJ*<, i.e. a subject whose accompanying verb is

[2]Note that comparison between languages is questionable, because of different language characteristics, parsing methods, annotation schemes, test data, evaluation methods, etc.

[3]By converting curly brackets (used to indicate syntactic functions), in the output of *IceParser*, to ordinary brackets, we were able to use the *Evalb* program for measuring the accuracy of the syntactic function annotation.

to the right, and an object whose accompanying verb is to the left. This was to be expected because the normal word order is SVO.

The F-measure for *COMP*< is only 75.1%. This can be explained by the relatively low recall (66.7%) as demonstrated in Section 8.3.2.

When *IceTagger* is used to produce tags, the overall F-measure for syntactic functions drops from 84.3% to 75.3% (see column 7 in table 8.2), which is equivalent to about 10.7% reduction in accuracy. Thus, the accuracy of the syntactic functions module is more sensitive to tagging errors than the constituent module. This can be explained by the fact that the former component relies to a much higher extent on the case feature, which is often responsible for the errors made by the tagger. Moreover, the additional errors, made by the phrase structure module when using POS tags from *IceTagger*, propagate through to the syntactic functions module.

Again, we are not in a position to compare our results to another Icelandic parser. For German (a related language), Müller (2004), for example, has presented the following results of syntactic function annotation using a finite-state parser (and correct POS tags from a corpus): 82.5% F-measure for all functions, and 90.8%, 64.5% and 81.9%, for subjects, accusative objects and dative objects, respectively. If these results are used for comparison, *IceParser* seems to obtain good results for syntactic functions.

Table 8.3 shows the accuracy of *IceParser* when the relative position indicator is ignored (many shallow parser do not include such an indicator). For example, when the three subject functions *SUBJ*, *SUBJ*> and *SUBJ*< are combined into one (*SUBJ*), the F-measure for *SUBJ* is 90.5%. Correspondingly, the F-measure for *OBJ* is 88.2%, when the three object functions *OBJ*, *OBJ*> and *OBJ*< are combined into one.

Due to the modular architecture of *IceParser*, we were able to feed our gold standard phrase structure annotation into the syntactic functions module with the purpose of getting upper bound figures for current version of the module. The results are shown in Table 8.4. The overall F-measure is 88.3%, which is 4.0% higher than the corresponding F-measure shown in Table 8.2. This demonstrates that it is important to try to increase the accuracy of the phrase structure module.

In the first version of *IceParser*, the output file of one transducer is used as an input file in the next transducer in the sequence. This version processes about 6,700 word-tag pairs per second (running on a Dell Optiplex GX620 Pentium 4, 3.20 GHz). We have implemented another version of the parser which, instead of reading and writing to files, reads from and writes directly

	Using correct POS tags			Using *IceTagger*			
Function type	Precision	Recall	F-measure	Precision	Recall	F-measure	Freq. in test data
SUBJ	56.1%	87.1%	68.2%	37.7%	64.7%	47.6%	4.7%
SUBJ>	95.5%	90.1%	92.7%	92.5%	86.4%	89.4%	30.3%
SUBJ<	86.5%	81.1%	83.7%	77.8%	72.5%	75.1%	12.3%
OBJ	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.2%
OBJ>	55.6%	35.7%	43.5%	33.3%	14.3%	20.0%	0.8%
OBJ<	91.1%	89.3%	90.2%	79.9%	76.6%	78.2%	19.7%
OBJAP>	60.0%	100.0%	75.0%	50.0%	66.7%	57.2%	0.2%
OBJAP<	83.3%	62.5%	71.4%	60.0%	37.5%	46.2%	0.4%
OBJNOM<	100.0%	18.2%	30.8%	100.0%	9.1%	16.7%	0.6%
IOBJ<	78.6%	68.8%	73.3%	63.6%	43.8%	51.9%	0.9%
COMP	44.8%	78.0%	56.9%	31.8%	54.0%	40.0%	2.8%
COMP>	91.3%	91.3%	91.3%	85.0%	73.9%	79.1%	1.3%
COMP<	85.9%	66.7%	75.1%	80.6%	91.8%	70.0%	12.7%
QUAL	85.7%	89.8%	87.7%	76.7%	79.4%	77.9%	10.4%
TIMEX	88.6%	64.6%	74.7%	95.0%	39.6%	55.9%	2.7%
All	85.3%	83.3%	84.3%	77.0%	73.7%	75.3%	100.0%

Table 8.2: Accuracy for the various syntactic functions

	Using correct POS tags			Using *IceTagger*		
Function type	Precision	Recall	F-measure	Precision	Recall	F-measure
SUBJ	90.4%	90.6%	90.5%	84.0%	85.0%	84.5%
OBJ	90.2%	86.3%	88.2%	79.1%	73.4%	76.2%
OBJAP	72.7%	72.7%	72.7%	55.6%	45.5%	50.0%
OBJNOM	100.0%	18.2%	30.8%	100.0%	9.1%	16.7%
IOBJ	78.6%	68.8%	73.3%	63.6%	43.8%	51.9%
COMP	83.3%	79.4%	81.3%	73.6%	68.4%	70.9%
QUAL	85.7%	89.8%	87.7%	76.7%	79.4%	77.9%
TIMEX	88.6%	64.6%	74.7%	95.0%	39.6%	55.9%
All	88.4%	86.3%	87.4%	80.4%	76.9%	78.6%

Table 8.3: Accuracy for the various syntactic functions when ignoring the position indicator

	Using correct POS tags		
Function type	Precision	Recall	F-measure
SUBJ	58.2%	91.8%	71.2%
SUBJ>	96.7%	91.2%	93.9%
SUBJ<	92.2%	84.7%	88.3%
OBJ	0.0%	0.0%	0.0%
OBJ>	55.6%	35.7%	43.5%
OBJ<	94.8%	92.7%	93.7%
OBJAP>	60.0%	100.0%	75.0%
OBJAP<	100.0%	87.5%	93.3%
OBJNOM<	100.0%	18.2%	30.8%
IOBJ<	78.6%	68.8%	73.3%
COMP	48.9%	86.0%	62.3%
COMP>	91.3%	91.3%	91.3%
COMP<	92.3%	75.0%	83.0%
QUAL	94.3%	97.3%	95.8%
TIMEX	97.0%	66.7%	79.0%
All	89.3%	87.2%	88.3%

Table 8.4: Accuracy of syntactic function annotation over gold-standard phrase bracketing

to memory (using the Java classes *StringReader* and *StringWriter*). This version annotates about 11,300 word-tag pairs per second, which is equivalent to about 75% speed increase compared to the previous version.

8.3 Error analysis

In this section, we analyse some of the errors made by *IceParser*, for the case of correct POS tags. First, we look at errors made during phrase annotation and, then, we consider errors in syntactic functions annotation. Lastly, we investigate the performance of *IceParser* when annotating special types of sentences, namely interrogative sentences and relative clauses. In the discussion, we occasionally refer to the individual transducers of *IceParser*, which are described in detail in Appendix C.

8.3.1 Errors in phrase annotation

Here, we show examples of phrase annotation errors for each of the different phrase types shown in table 8.1, for which the F-measure is below 99.0%.

- **AdvP**
  The only type of error in adverb phrase annotation occurs when the parser incorrectly groups together two (or more) adjacent adverbs. Consider the incorrect output (in the examples below, the POS tag follows each word):

  [PP um ao [NP það fpheo NP] PP] [VP vissi sfg3eþ VP] [NP stelpan nveng NP] [AdvP ekki aa þá aa AdvP]
  (about that knew girl not then)

  The two adverbs at the end should form two distinct AdvPs because *ekki* is a sentence adverb which does not modify the temporal adverb *þá*.

- **AP**
  Adverbs are also the source of some of the errors made in then annotation of adjective phrases. Consider the incorrect output:

  [CP og CP] [VP tóku sfg3fþ VP] [NP [AP [AdvP fram aa AdvP] eigin lvfoof AP] dósir nvfo NP]
  (and took out own cans)

  In this sentence part, the adverb *fram* is a particle associated with the verb *tóku* (take out), but not a modifier of the adjective *eigin*. This type of error could be reduced by building an exhaustive list of verb-particle pairs[4].

- **APs**
  Since an APs consists of a sequence of adjective phrases, errors in the latter phrases affect the former. Consider, for example, the incorrect

[4]Note that, for this purpose, we would need all wordforms in singular and plural, present and past tense, indicative and subjunctive mood, etc., for the given verbs. Arguably, this list would be quite large.

output:

[VP stóð sfg3eþ VP] [NP hann fpken NP] [APs [AP [AdvP uppi aa AdvP] slyppur lkensf AP] [CP og c CP] [AP snauður lkensf AP] APs]
(stood he up empty-handed and poor)

Once again, an adverb is the root of the error — *uppi* is a particle associated with the verb *stóð*, but not a modifier of the adjective *slyppur*. The adjective phrase *uppi slyppur* is therefore wrong, and, consequently, the whole sequence of adjective phrases is incorrect according to the evaluation method.

To illustrate another type of error in APs annotation, consider the following incorrect output:

[NP jarðvegurinn nkeng NP] [VPb er sfg3en VPb] [APs [AP frjósamur lkensf AP] [CP og c CP] [AP gljáandi lkenof AP] , , [AP tilbúinn lkensf AP] APs] [VPi að cn gefa sng VPi]
(the soil is fertile and shiny, ready to give)

An APs is defined as two or more adjective phrases, separated by a conjunction or a comma (and agreeing in gender, number and case). This pattern applies to the substring above and, therefore, *IceParser* groups the three adjective phrases into one APs. The last adjective, *tilbúinn*, should, however, not be a part of the APs because it is not part of the enumeration. This type of error could be fixed by modifying the definition of a APs, i.e. by not allowing a comma to appear after a conjunction has appeared.

- **NP**
  A frequent noun phrase error made by the parser is exemplified by the incorrect output:

  [NP árin nhfng NP] [AP gullnu lhfnvf AP]
  (years golden)

  Here, the correct annotation is *[NP árin nhfng [AP gullnu lhfnvf AP] NP]* because the adjective *gullnu* is a post-modifier of the noun *árin*.

IceParser makes this type of error because it does not include a pattern for <noun/pronoun-adjective> word order.

Note that this type of error can not simply be accounted for by adding a regular expression to the *Phrase_NP* transducer, which groups a noun/pronoun and a following adjective, agreeing in gender, number and case, together into a NP. The reason is that, even though an agreement holds between a noun/pronoun and a following adjective, the two words might not be a part of the same NP! To see why, consider the following sentence part:

[VPb er sfg3en VPb] [NP hann fpken NP] [AP ólíkur lkensf AP]
(is he different)

Here, feature agreement holds between the pronoun *hann* and the adjective *ólíkur* (as can be seen by comparing the corresponding POS tags), but these words constitute two separate noun phrases, corresponding to a subject and a verb complement.

When the word order <adjective-noun> occurs in Icelandic text, it is very likely that the words are part of the same noun phrase, i.e. that the adjective modifies the noun. There are exceptions to this — consider, for example, the incorrect output:

[NP þau fphfn NP] [VPb voru sfg3fþ VPb] [NP [AP barnlaus lhfnsf AP] hjónin nhfng NP]
(they were childless couple-the)

In this case, the adjective *barnlaus* is not a modifier but a complement of the verb *voru*. The meaning of the sentence is essentially: *they couple-the were childless*, and, therefore, the correct annotation is:

[NP þau fphfn NP] [VPb voru sfg3fþ VPb] [AP barnlaus lhfnsf AP] [NP hjónin nhfng NP]

An extreme example of an error made by *IceParser* for noun phrase annotation is the following output:

[NP einna fohfe [AP líkast lhense AP] kló nveþ NP]

(rather like plug)

These words are grouped into a noun phrase because the pattern <indefinite pronoun, adjective, noun> is a normal word order in such phrases. In this case, however, all the three words have different case markers, and should therefore be annotated as:

[NP einna fohfe NP] [AP líkast lhense AP] [NP kló nveþ NP]

This type of error can be eliminated by using the case feature in the patterns of the *Phrase_NP transducer*.

- **NPs**
  The *Phrase_NPs* transducer groups together a sequence of noun phrases agreeing in case. Such a sequence, typically, denotes an enumeration of some kind (see Section 6.2.1.8). This grouping can, however, result in errors. Consider, for example, the incorrect output:

  [AP sterkur lkensf AP] [VPb var sfg3eþ VPb] [NPs [NP hann fpken NP] [CP og c CP] [NP íþróttamaður nken NP] NPs] [AP ágætur lkensf AP]
  (strong was he and athlete fine)

  Here, the parser groups the noun phrases *[NP hann fpken NP] and [NP íþróttamaður nken NP]* together because the phrases agree in case. The correct annotation, however, is:

  [AP sterkur AP] [VPb var VPb] [NP hann NP] [CP og CP] [NP íþróttamaður [AP ágætur AP] NP]

  In general, the regular expressions, used by the *Phrase_NPs* transducer, demand the comma token or a conjunction between the noun phrases. Thus, noun phrases like:

  [NPs [NP orðið nheng NP] [NP hjátrú nven NP] NPs]
  (word-the superstition)

. . . are incorrectly annotated by *IceParser* as:

[NP orðið nheng NP] [NP hjátrú nven NP]

On the other hand, the *Phrase_NPs* transducer does not demand a token between two noun phrases, when the former phrase consists of a proper noun and the latter of a common noun, for example:

[NPs [NP Jónas nken-m NP] [NP skólastjóri nken NP] NPs]
(Jónas headmaster)

This is quite a common scenario, but, the corresponding pattern can result in errors like:

[NPs [NP Jóna nven-m NP] [NP mamma nven NP] NPs] [NPs [NP hans fpkee NP] [NP Óla nkee-m NP] NPs]
(Jóna mother his Óla)

In this case, the correct annotation is:

[NP Jóna nven-m NP] [NP mamma nven NP] [NPs [NP hans fpkee NP] [NP Óla nkee-m NP] NPs]

- **PP**
  Prepositions in Icelandic are the only type of adpositions. However, a certain few prepositions can, in fact, appear after the associated noun phrase. The *Phrase_PP* transducer of *IceParser* does contain patterns for some of these few exceptions, but not all of them. For example, *IceParser* generated the incorrect output:

  [VP fóru sfg3fþ VP] [NP fjandans nkeeg NP] [PP til ae PP]
  (went devil to)

  This is, in fact, an idiom, and should be annotated as:

  [VP fóru sfg3fþ VP] [PP [NP fjandans nkeeg NP] til ae PP]

Since a NP or a NPs is included in a PP, errors in the annotation of the former phrase result in annotation errors in the latter phrase. To give on example, consider the incorrect output:

[PP á aþ [NP jörðu nveþ NP] PP] [SCP sem c SCP] [NP himni nkeþ NP]
(on earth as in heaven)

Here, the two noun phrases agree in case and are separated by a subordinating conjunction, which, in fact, functions as a coordinating conjunction. The correct annotation is thus:

[PP á aþ [NPs [NP jörðu nveþ NP] [SCP sem c SCP] [NP himni nkeþ NP] NPs] PP]

The *Phrase_NPs* transducer, however, only considers coordinating conjunctions when grouping noun phrases, and the above therefore results in a PP annotation error.

Errors in the annotation of a special kind of MWE, the *MWE_PP*, can result in errors in annotation of preposition phrases. This is demonstrated in the next item below.

- **MWE**
  IceParser relies on a list of multiword expressions for each of the different kinds of MWEs (see Appendix C.4). This list was used when hand-annotating the *gold standard*, and, therefore, the recall for MWEs is 100% (see table 8.1). On the other hand, precision is not 100% because, in some cases, the MWEs found by *IceParser* are actually not MWEs in the particular context. Consider, for example, the following incorrect output of *IceParser*:

  [NP ég fp1en NP] [VP stökk sfg1eþ VP] [PP [MWE_PP út aa við ao MWE_PP] [NP kirkjugarðinn nkeog NP] PP]
  (I jumped out by cemetery-the)

  IceParser annotates the adverb-preposition pair *út-við* (out-by) as a MWE, but in this case the adverb *út* is actually a particle associated

with the preceding verb *stökk* (jumped). As pointed out above, such type of errors could be reduced by a list of verb-particle pairs.

8.3.2 Errors in syntactic functions annotation

As can be deduced from tables 8.1 and 8.2, the syntactic functions module is less reliable than the phrase structure module. The reason is twofold. First, as discussed in Chapter 3, Icelandic word order is rather free, and this freedom mainly concerns the relative order of major syntactic constituents, such as noun phrases (subjects and objects), preposition phrases and adverb phrases. Second, phrase annotation is needed for the annotation of syntactic functions, and, hence, errors in the former can produce errors in the latter.

In this section, we analyse some of the errors made by *IceParser* when annotating syntactic functions.

- **Subjects**
  To give an example where *IceParser* annotates a subject without a correct position indicator, consider the output:

  *[VPb er sfg3en VPb] [AdvP ekki aa AdvP] [VPi að cn koma sng VPi] {*SUBJ [NP matur nken NP] *SUBJ} ? ?*
  (is not to come food?)

  The correct annotation for the subject is *{*SUBJ< [NP matur NP] *SUBJ<}* because *matur* is the subject of the verb *er* at the beginning of the sentence. *IceParser* does include patterns to match the order <VP-AdvP-SUBJ>, but, in this case, the infinitive verb phrase *[VPi að koma VPi]* is positioned in between the AdvP and the SUBJ. The parser, however, marks all stand-alone nominative noun phrases as *{*SUBJ ... *SUBJ}*, and therefore the *[NP matur NP]* phrase does receive subject marking, albeit incomplete.

  In some cases, the parser does not recognise a subject whose accompanying word is directly to the right. A typical example is the following incorrect annotation:

  *[NP honum fpkeþ NP] [VPb var sfg3eþ VPb] {*COMP< [AP ókunnugt lhensf AP] *COMP<} [PP um ao PP]*

(he was unaware of)

Here, the noun phrase should be marked as emph{*SUBJ> [NP honum fpkeþ NP] *SUBJ>} — the subject is an oblique case subject (i.e. a non-nominative subject). The pattern <oblique case subject - the verb *vera* - an adjective> is not uncommon, but the parser does not include patterns for recognising oblique case subjects associated with the verb *vera*.

- **Objects**
  As mentioned above, some of the errors in syntactic function annotation are due to errors made in the phrase structure annotation. For the incorrect phrase structure output (discussed in Section 8.3.1):

  [CP og CP] [VP tóku sfg3fþ VP] [NP [AP [AdvP fram aa AdvP] eigin lvfoof AP] dósir nvfo NP]
  (and took out own cans)

  ...the parser will produce the syntactic function:

  *{*OBJ< [NP [AP [AdvP fram aa AdvP] eigin lvfoof AP] dósir nvfo NP] *OBJ<}*

  This object is incorrect because it includes the adverb phrase *[AdvP fram aa AdvP]*.

  The F-measure for indirect objets (*{*IOBJ< ... *IOBJ<}*) is only about 73%. The reason is that a number of possible patterns for indirect objects are not included in the parser. For example, consider the incorrect output:

  *{*SUBJ> [NP þú fp2en NP] *SUBJ>} [VP getur sfg2en [AdvP ekki aa AdvP] unnt ssg VP] {*OBJ< [NP mér fp1eþ NP] {*QUAL [NP þess fphee NP] *QUAL} *OBJ<}*
  (you can not comply me that)
  (literally meaning "you cannot comply with me doing that")

Here, the di-transitive verb *unnt* demands an dative indirect object and a genitive direct object, resulting in the correct annotation:

*{*SUBJ> [NP þú fp2en NP] *SUBJ>} [VP getur sfg2en [AdvP ekki aa AdvP] unnt ssg VP] {*IOBJ< [NP mér fp1eþ NP] *IOBJ<} {*OBJ< [NP þess fphee NP] *OBJ<}*

IceParser, however, only accounts for a <dative,accusative> or <accusative,dative> pattern for indirect and direct objects, and, therefore, annotates the genitive object as a genitive qualifier.

The F-measure for objects of an adjective (*{*OBJAP< ... *OBJAP<}*, *{*OBJAP> ... *OBJAP>}*) is only about 70-75%. Recall (from Section 8.3.1) the incorrect output:

*{*OBJ< [NP einna fohfe [AP líkast lhense AP] kló nveþ NP] *OBJ<}*
(rather like plug)

This is an example of a phrase annotation error which results in errors in syntactic functions annotation. The correct annotation is:

*{*QUAL [NP einna fohfe NP] *QUAL} {*COMP< [AP líkast lhense AP] *COMP<} {*OBJAP< [NP kló nveþ NP] *OBJAP<}*

- **Verb complements**
  Adverbs are responsible for some of the errors in verb complement annotation. For example, *IceParser* generates the following incorrect output:

  *[AdvP samt aa AdvP] {*COMP [AP [AdvP stundum aa AdvP] æstur lkensf AP] *COMP}*
  (nevertheless sometimes upset)

  The adverb *stundum* does not modify the adjective *æstur*, and thus the correct annotation is:

*[AdvP samt aa AdvP] [AdvP stundum aa AdvP] {*COMP [AP æstur lkensf AP] *COMP}*

In our discussion on NP errors, we mentioned an error occurring when an adjective post-modifies a noun. This same error is partly to blame for the relatively low recall for verb complements. Consider again the noun phrase *[NP árin nhfng [AP gullnu lhfnvf AP] NP]*, which in our gold standard appears in the sentence:

*{*SUBJ> [NP þetta fahen NP] *SUBJ>} [VPb voru sfg3fþ VPb] {*COMP< [NP árin nhfng [AP gullnu lhfnvf AP] NP] *COMP<}*
(These were years golden)

Since this NP is incorrectly annotated by the *Phrase_NP* transducer, the *Func_COMP* transducer is not able to correctly annotate this complement, which in turn negatively affects the recall for **COMP<*.

In some cases, *IceParser* has problems distinguishing between subjects and verb complements. Consider, for example, the incorrect output:

*[AdvP kannski aa AdvP] [VPb hafði sfg3eþ [AdvP aðeins aa AdvP] verið ssg VPb] {*SUBJ< [NP kjánaskapur nken NP] *SUBJ<} [PP af aþ [NP honum fpkeþ NP] PP]*
(maybe had only been silliness of him)

Because of a missing subject before the verb *verið*, the parser assumes an inverted word order, i.e. that the nominative subject appears after the verb. The noun *kjánaskapur* is, however, a verb complement in this context, and the correct annotation is:

*[AdvP kannski aa AdvP] [VPb hafði sfg3eþ [AdvP aðeins aa AdvP] verið ssg VPb] {*COMP< [NP kjánaskapur nken NP] *COMP<} [PP af aþ [NP honum fpkeþ NP] PP]*

Another example demonstrating a subject/verb complement error is the following incorrect output:

*{*SUBJ> [NP það fphen NP] *SUBJ>} [VPb gátu sfg3fþ verið ssg VPb] {*COMP< [NP slitrur nvfn NP] *COMP<} [PP úr aþ [NP dag-*

*blaði nheþ NP] PP] , , {*SUBJ [NP sígarettustubbar nkfn NP] *SUBJ}*
*, , [AdvP jafnvel aa AdvP] {*SUBJ [NP plástrar nkfn NP] *SUBJ}*
(it could be slits from newspaper, cigarettes-stubs, even plasters)

Here, both *[NP sígarettustubbar nkfn NP]* and *[NP plástrar nkfn NP]* should be verb complements belonging to the preceding verb phrase *[VPb gátu sfg3fþ verið ssg VPb]*, but not subjects. Again, this type of error has a negative affect on the recall for **COMP<*.

- **Qualifiers**
  Consider the incorrect annotation:

  *{*QUAL [NP hvorugur foken drengjanna nkfeg NP] *QUAL}*
  (neither boys-the)

  Here, an incorrect annotation of the noun phrase, results in an error in the annotation of the genitive qualifier (and a missing annotation of a subject). The correct annotation is:

  *{*SUBJ> [NP hvorugur foken NP] *QUAL [NP drengjanna nkfeg NP] *QUAL *SUBJ>}*

8.3.3 Interrogative sentences and relative clauses

In Section 8.3.2 we analysed some errors in the output of *IceParser* with regard to syntactic functions. Some of these errors point to the limitations of the shallow (local) parsing mechanism. In this section, we investigate the syntactic function annotation of *IceParser* with regard to interrogative sentences and relative clauses. We would like to see if these type of constructions are in some way more problematic for our parser.

In Icelandic (and in many other Germanic languages) an interrogative sentence normally changes the word order so that the verb phrase appears before the subject. However, the verb-subject-object order is not constrained to interrogative sentences in Icelandic (as exemplified in Section 6.2.2.2). Therefore the general patterns for handling subjects in declarative sentences should work equally well for interrogative sentences.

Consider the following examples[5]:

*[VP gekk sfg3eþ VP] {*SUBJ< [NP ferðin nveng NP] *SUBJ<} [AdvP ekki aa vel aa AdvP] ?*
(went trip-the not well?)

*[VPb eru sfg3fn VPb] {*SUBJ< [NP þær fpvfn NP] *SUBJ<} {*COMP< [AP [AdvP ekki aa AdvP] unaðslegar lvfnsf AP] *COMP<} ?*
(are they not lovely?)

*[AdvP hvar aa AdvP] [VPb er sfg3en VPb] {*SUBJ< [NP þetta fahen NP] *SUBJ<} ?*
(where is this?)

In all these three examples the subject is correctly annotated and the relative position indicator points to the verb phrase appearing to the left of the subject.

Of course, *IceParser* does not annotate all interrogative sentences correctly. Consider the following output:

*{*SUBJ> [NP hvað fshen NP] *SUBJ>} [VPb eru sfg3fn VPb] {*SUBJ [NP þeir fpkfn NP] *SUBJ} {*COMP [AP margir lkfnsf AP] *COMP} ?*
(how are they many?)

In this example, the noun phrase *[NP hvað fshen NP]* is not the subject of the verb *eru*, but rather the noun phrase *[NP þeir fpkfn NP]*. The *Func_SUBJ* transducer annotates the first noun phrase as the subject, because the word *hvað* is in the nominative case and appears to the left of a finite verb phrase. The second noun phrase is then (in a later transducer) annotated as a subject without a relative position indicator, because it is considered a stand-alone nominative noun phrase. For a similar reason, the adjective phrase at the end of the sentence is annotated as a stand-alone complement.

The correct annotation is:

*[NP hvað fshen NP] [VPb eru sfg3fn VPb] {*SUBJ< [NP þeir fpkfn NP] *SUBJ<} {*COMP< [AP margir lkfnsf AP] *COMP<} ?*

[5] All examples in this section are taken from the output of *IceParser*.

(how are they many?)

Let us now consider relative clauses. The only rule with regard to relative clauses in *IceParser* is to mark nominative subjects appearing to the left of a relativizer (the word *sem* tagged as *ct*)[6]. This works reasonably well, as can be seen in the following examples:

*{*SUBJ> [NP ég fp1en NP] *SUBJ>} [VP reyndi sfg1eþ VP] {*SUBJ> [NP allt fohen NP] *SUBJ>} [SCP sem ct SCP] {*COMP> [AP hægt lhensf AP] *COMP>} [VPb var sfg3eþ VPb]*
(I tried everything that possible was)

The noun phrase *[NP allt fohen NP]* is correctly annotated as the subject of the verb *var* in the relative clause.

As another illustration, consider the following output:

*{*SUBJ> [NP [AP ungur lkensf AP] maður nken NP] *SUBJ>} [SCP sem ct SCP] [VP ákvað sfg3eþ VP] [VPi að cn bjarga sng VPi] {*OBJ< [NP jörðinni nveþg NP] *OBJ<}*
(young man who decided to save earth-the)

Here the noun phrase *[NP [AP ungur lkensf AP] maður nken NP]* is correctly annotated as the subject of the verb *ákvað* in the relative clause.

As an example of an error in the context of a relative clause, consider the following:

*[CP og c CP] {*SUBJ [NP þeir fakfn menn nkfn NP] *SUBJ} [PP í aþ [NP [AP mestum lhfþsf AP] metum nhfþ NP] PP] [SCP sem ct SCP] {*COMP> [AP skæðastir lkfnse AP] *COMP>} [VPb eru sfg3fn VPb]*
(and those men in most respected that vicious are)

In this example, a prepositional phrase appears between the noun phrase *[NP þeir fakfn menn nkfn NP]* and the relativizer. *IceParser* does not account for this possibility when marking a subject and therefore this particular

[6] A relativizer is a subordinating conjunction that links a relative clause to its head noun.

subject has a missing relative position indicator pointing to the right.

Recall (from Section 3.2) that the case feature in the POS tags provides an important clue regarding the syntactic function of a phrase. For a simple sentence like *Maðurinn elskar Maríu* (Man-the loves Mary) the parser returns the correct output (*Maðurinn* is in the nominative case and *Maríu* is in the accusative case):

*{*SUBJ> [NP Maðurinn nkeng NP] *SUBJ>} [VP elskar sfg3en VP] {*OBJ< [NP Maríu nveo-m NP] *OBJ<}*

The case feature is also valuable when annotating syntactic functions in relative clauses. Consider the following output:

*{*SUBJ> [NP Maðurinn nkeng NP] *SUBJ>} [VP elskar sfg3en VP] {*OBJ< [NP Maríu nveo-m NP] *OBJ<} [SCP sem ct SCP] [VP keypti sfg3eþ VP] {*OBJ< [NP hest nkeo NP] *OBJ<}*
(Man-the loves Mary who bought (a) horse)
(In case this English gloss is ambiguous, it is Mary who bought a horse)

Again this is a correct annotation (according to our annotation scheme). The noun phrase *[NP Maríu nveo-m NP]* is in the accusative case and is therefore not a subject candidate.

Furthermore, consider the following sentence:

*{*SUBJ> [NP Maðurinn nkeng NP] *SUBJ>} [SCP sem ct SCP] [VP elskar sfg3en VP] {*OBJ< [NP Maríu nveo-m NP] *OBJ<} [VP keypti sfg3eþ VP] {*OBJ< [NP hest nkeo NP] *OBJ<}*
(Man-the who loves Mary bought (a) horse)
(In case this English gloss is ambiguous, it is the man who bought a horse)

In examples like this one, *IceParser* is not "tempted" to annotate the noun phrase in the embedded clause (*[NP Maríu nveo-m NP]*) as the subject of the verb in the main clause (*keypti*), because the noun phrase is in the accusative case. The parser thus correctly marks the noun phrase as an object of the verb *elskar*.

These examples demonstrate that the parser consistently annotates subjects and objects in sentences with or without a relative clause.

On the other hand, *IceParser* has problems annotating similar sentences

where the main verb demands a non-nominative subject (in such cases the object is in the nominative case). To illustrate, consider the following output:

*[NP Manninum nkeþg NP] [SCP sem ct SCP] [VP áskotnaðist sfm3eþ VP] {*SUBJ< [NP [AP mikill lkensf AP] auður nken NP] *SUBJ<} [VP keypti sfg3eþ VP] {*OBJ< [NP hest nkeo NP] *OBJ<}*
(man-the who acquired great wealth bought (a) horse)

Here the verb *áskotnaðist* demands a non-nominative subject (note that the noun *manninum* is in the dative case) and a nominative object. In this case, the parser incorrectly marks the noun phrase *[NP [AP mikill lkensf AP] auður nken NP]* as the subject of the verb *áskotnaðist*. The correct annotation is:

*{*SUBJ> [NP Manninum nkeþg NP] *SUBJ>} [SCP sem ct SCP] [VP áskotnaðist sfm3eþ VP] {*OBJNOM< [NP [AP mikill lkensf AP] auður nken NP] *OBJNOM<} [VP keypti sfg3eþ VP] {*OBJ< [NP hest nkeo NP] *OBJ<}*

The following demonstrates a similar type of an error with regard to a relative clause and a dative subject:

*[VP hafði sfg3eþ VP] [AdvP ekki aa AdvP] {*OBJ< [NP augun nhfog NP] *OBJ<} [PP af aþ [NP manninum nkeþg NP] PP] [SCP sem ct SCP] [VP skolaði sfg3eþ VP] {*SUBJ< [NP ólífunum nvfþg NP] *SUBJ<} [PP [MWE_PP niður aa með aþ MWE_PP] [NP viskíinu nheþg NP] PP]*
(had not eyes off man-the who washed olives-the down with whisky-the)

The correct subject of the verb *skolaði* (washed) is the noun phrase *[NP manninum nkeþg NP]* which is a part of a preposition phrase[7]. *IceParser* incorrectly marks the dative noun phrase *[NP ólífunum nvfþg NP]* as a subject of the verb *skolaði*, because this particular verb can take a dative subject! In that case, the verb has a different meaning, e.g. *líkinu skolaði á land*, meaning "the body washed (from the sea) to the shore".

IceParser contains regular expressions for recognising non-nominative

[7] *IceParser* never marks a noun phrase contained in a preposition phrase as a subject – it is simply not part of the annotation scheme.

subjects[8], but these expressions do not handle embedded clauses like the ones above.

In this section, we have illustrated that *IceParser* annotates syntactic functions reasonably accurately for interrogative sentences and relative clauses. On the other hand, we have also showed that there exist frequent sentence constructions which our parser does not annotated correctly.

8.4 Conclusion

In this chapter, we have evaluated the incremental finite-state parser *IceParser*, for parsing Icelandic text. The parser comprises two modules: the phrase structure module and the syntactic functions module. Both modules consist of a sequence of transducers, which add syntactic information into the input strings, according to our shallow syntactic annotation scheme.

Evaluation shows that F-measure for phrases and syntactic functions is 96.7% and 84.3%, respectively, when assuming correct POS tagging. We have argued that these results are good because Icelandic has a relatively free word order, which is difficult to account for in a parser.

Only three out of eleven phrase types result in less than 95% accuracy, and for the three most common phrase types (NP, PP, and VPx) the F-measure is about 97% or higher. Error analysis shows that adverbs are a frequent source of errors made in the annotation of phrase structure. We have indicated that the errors associated with adverbs could be reduced by including a list of frequent verb-particle pairs in the parser. Moreover, the noun phrase transducer only relies on word order, but by adding gender, number and case agreement into its patterns, its accuracy would probably increase.

For the two most common syntactic functions (**SUBJ>* and **OBJ<*), the F-measure is above 90%. By ignoring the position indicator in the function labels, the F-measure for syntactic function increases from 84.3% to 87.4%. Since the syntactic function annotation depends on the phrase annotation, it is very likely that improving the latter component will result in higher accuracy in the former component.

Not surprisingly, our evaluation shows that accuracy of the parser decreases when the POS tags are not perfect. When *IceTagger* is used for tagging the

[8] These patterns are recognised with the help of a list of verbs demanding such subjects.

test data, we noticed a 5% reduction in accuracy for phrase structure, and 10.7% reduction in accuracy for syntactic functions. This indicates that it is of prime importance to further improve the tagging accuracy of Icelandic text.

We demonstrated that the overall upper bound on F-measure for the current version of the syntactic functions module is 88.3%, obtained by running the gold standard phrase annotation through the syntactic functions module.

Our parsing results are the first published results for the Icelandic language. We have, therefore, not been able to compare our results to other parsers for Icelandic. On the other hand, a comparison with related languages indicates that our results are good.

In Section 6.1, we hypothesised that the use of a finite-state parsing method for a morphologically complex language, with a relatively free word order, like Icelandic, is effective, and additionally, that an enormous effort is not needed in the development of a finite-state parser for the language in order to obtain good results. We believe that our evaluation shows that finite-state parsing methods are, indeed, effective for Icelandic. Moreover, good results were obtained by using only about 1 man year in development, including the construction of the GDC, the development and testing of the parser, the construction of the gold standard and evaluation work.

In Section 6.1, we also mentioned the importance of the case feature for the purpose of grouping words together into phrases and identifying syntactic functions. This feature has been used when writing many of the syntactic patterns in *IceTagger*, but our error analysis has showed the benefit of using this feature in more patterns.

In future work, we would like to improve individual components of our parser, and build a version of it which utilises to a greater extent the morphological information available in the POS tags. Moreover, it would be interesting to use the shallow output of *IceParser* as input into a full (deep) parsing system.

Part IV

Concluding remarks

Chapter 9

Conclusions

In this last chapter, we give a summary of the thesis and some directions for future work.

In this thesis, we investigated the effectiveness and viability of using (mainly) rule-based methods for analysing the syntax of Icelandic text. We developed a Natural Language Processing Toolkit, *IceNLP*, for this purpose. The toolkit consists of a tokeniser/sentence segmentiser, the morphological analyser *IceMorphy*, the linguistic rule-based POS tagger *IceTagger*, and the shallow parser *IceParser*. In addition to developing a new tagger, we experimented with integration methods and combination methods of various taggers, for the purpose of increasing the tagging accuracy. The above tools and methods have been tested and evaluated, and compared to other applicable methods.

The main motivation for our work was the lack of basic tools for processing the Icelandic language, and we argued that this work is a step towards the goal of developing a *BLARK* for the language.

The first part of the thesis (Part I – Introduction) presented the motivation for our work, and introduced background material on POS tagging and syntactic analysis. Additionally, in this first part, we briefly described the main characteristic of the Icelandic language.

In the second part (Part II – Data and System), we, first, described the data used for development of the tools, as well as for evaluation. Secondly, we described the tagging system, including tokenisation, the morphological analyser, *IceTagger*, and *TriTagger*, our re-implementation of a known statistical tagger. Lastly, the parsing system was described, including our shallow syntactic annotation scheme and the functionality of our finite-state parser,

IceParser

Evaluation of the system along with error analysis is presented in the third part (Part III – Evaluation). We evaluated individual components of *IceTagger*, i.e. the morphological analyser/unknown word guesser, the heuristics, and the performance of the tagger as a whole. Overall, *IceTagger* achieves about 91.5% tagging accuracy when tested using the *IFD* corpus. This is about 1.1% percentage points higher than achieved by the best data-driven tagger. Moreover, we presented evaluation results using other corpora than *IFD*, which seem to indicate that our tagger is less sensitive to different test material than the data-driven taggers.

We, originally, hypothesised that higher tagging accuracy could be obtained by developing a linguistic rule-based tagger (without using an enormous effort in the development) than achieved by a state-of-the-art statistical tagger. Our tagging system was developed in only 7 man months, which can be considered a short development time for a linguistic rule-based system.

By using various integration and combination methods, we were able to increase the tagging accuracy of Icelandic text. Our best integrated tagger, consisting of *IceTagger* and *TriTagger*, achieves 91.8% accuracy. By combining 5 different taggers, using a simple voting scheme and two linguistically motivated rules, we obtained an accuracy of about 93.5%.

Both modules of *IceParser*, i.e. the phrase structure module and the syntactic functions module, were evaluated. The former module achieves 96.7% F-measure, and the same measure for the latter module is 84.3%. These results, which are comparable to results for related languages, are obtained by using only a limited amount of information available in the POS tags for each word. The overall time spent on development of the parsing system was about 1 man year. Our work shows that finite-state parsing methods are both effective and viable for a morphologically complex language with a relatively free word order, like Icelandic. Note, moreover, that no treebank exists for Icelandic, and, thus, using data-driven parsing methods is currently not an option.

9.1 Future work

In the first part of this thesis, we argued that language technology has just taken its first steps in Iceland. Consequently, a substantial effort, in the coming years, will be devoted to further developing the BLARK for the

Icelandic language. Tools like a tagger and a parser are now a part of the BLARK, and we like to consider our work in this area as a foundation for further research and development.

In the evaluation part of this work, we have discussed a number of issues that we would like to work on in the future — here we summarise those:

- By using an extensive dictionary (instead of a dictionary derived from the *IFD* corpus) the precision and recall of *IceMorphy* might improve.
- By writing more local rules in *IceTagger* to handle frequent ambiguous word forms, the accuracy of the tagger should improve.
- The heuristics used by *IceTagger* can be improved, which should result in higher tagging accuracy.
- The main dictionary used by *IceTagger* contains less than 60,000 word forms. Intuitively, using larger (and more comprehensive) dictionaries (derived from the large resource *Morphological Description of Icelandic*) should result in higher accuracy because of fewer occurrences of unknown words. However, larger dictionaries could result in higher average ambiguity rate, which generally reduces the tagging accuracy. Therefore, experimenting with different dictionary sizes is an interesting project.
- The tagset used in this research is large, i.e. consists of about 660 tags. An important future work is to design a smaller version of this tagset and evaluate the taggers using this new tagset. When designing the smaller tagset, the main decision is what features of the large tagset can be left out without to much loss of information.
- Evaluating the data-driven taggers using a larger tagged corpus (i.e. larger than *IFD*) is now a feasible option. The best performing combination tagger can be used initially to tag new text, followed by hand-correction.
- In our experiments with combination of taggers, we have only used voting methods. We would like to experiment with stacking methods, e.g. methods based on memory-based learning or maximum entropy.
- It is worthwhile to improve individual components of our parser, *IceParser*.

- An interesting alternative is to build a version of the parser, which utilises to a greater extent the morphological information available in the POS tags.
- The shallow output of our parser could be used as an input into a full (deep) parsing system.

Bibliography

S. Abney. Parsing by Chunks. In R. Berwick, S. Abney, and C. Tenny, editors, *Principle-Based Parsing*. Kluwer Academic Publishers, 1991.

S. Abney. 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, 1996.

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

A. Aho, M. Lam, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques and Tools*. Addison Wesley, 2007.

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

S. Aït-Mokhtar, J.-P. Chanod, and C. Roux. Robustness beyond Shallowness: Incremental Deep Parsing. *Natural Language Engineering*, 8(3):121–144, 2002.

M. Albertsdóttir and S. Stefánsson. Beygingar- og málfræðigreinikerfi. In *Samspil tungu og tækni – Afrakstur tungutækniverkefnis menntamálaráðuneytisins*. Menntamálaráðuneytið, Reykjavik, Iceland, 2004.

A. Arnalds. Language Technology in Iceland. In H. Holmboe, editor, *Nordisk Sprogteknologi 2003*. Museum Tusculanums Forlag, Copenhagen, 2003.

J. Birn. Swedish Constraint Grammar: A short presentation. Technical report, Lingsoft Inc., Finland, 1998.

K. Bjarnadóttir. Modern Icelandic Inflections. In H. Holmboe, editor, *Nordisk Sprogteknologi 2005*. Museum Tusculanums Forlag, Copenhagen, 2005.

E. Black, S. Abney, D. Flickinger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. A Procedure for Quantitatively Comparing the Syntactic Coverage of English Grammars. In *Proceedings of the DARPA Speech and Natural Language Workshop*, Pacific Grove, CA, USA, 1991.

L. Borin. Something borrowed, something blue: Rule-based combination of POS taggers. In *Proceedings of the* 2^{nd} *International Conference on Language Resources and Evaluation*, Athens, Greece, 2000.

T. Brants. TnT: A statistical part-of-speech tagger. In *Proceedings of the* 6^{th} *Conference on Applied Natural Language Processing*, Seattle, WA, USA, 2000.

S. Briem. Automatisk morfologisk analyse af islandsk tekst. In *Papers from the Seventh Scandinavian Conference of Computational Linguistics*, Reykjavik, Iceland, 1989.

E. Brill. A Simple Rule-Based Part of Speech Tagger. In *Proceedings of the* 3^{rd} *Conference on Applied Natural Language Processing*, Trento, Italy, 1992.

E. Brill. Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging. *Computational Linguistics*, 21(4):543–565, 1995a.

E. Brill. Unsupervised Learning of Disambiguation Rules for Part of Speech Tagging. In *Proceedings of the* 3^{rd} *Workshop on Very Large Corpora*, Cambridge, MA, USA, 1995b.

E. Brill and J. Wu. Classifier Combination for Improved Lexical Disambiguation. In *COLING-ACL '98:* 36^{th} *Annual Meeting of the Association for Computational Linguistics and* 17^{th} *International Conference on Computational Linguistics*, Montreal, Quebec, Canada, 1998.

J. Carroll, T. Briscoe, N. Calzolari, S. Federice, S. Montemagni, V. Pirrelli, G. Grefenstette, A. Sanfilippo, and G. Carroll. SPARKLE Work Package 1:

Specification of Phrasal Parsing. Technical Report LE1-2111, Commission of the EC, Telematics Applications Programme, Language Engineering, 1997.

J. Carroll, E. Briscoe, and A. Sanfilippo. Parser evaluation: a survey and a new proposal. In *Proceedings of the* 1^{st} *International Conference on Language Resources and Evaluation*, Granada, Spain, 1998.

J.-P. Chanod and P. Tapanainen. Tagging French – comparing a statistical and a constraint-based method. In *Proceedings of the* 7^{th} *Conference on European Chapter of the ACL (EACL)*, Dublin, Ireland, 1995.

J.-P. Chanod and P. Tapanainen. A robust finite-state parser for French. Technical report, Xerox Corporation, 1996.

E. Charniak. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the* 14^{th} *National Conference on Artificial Intelligence, AAAI*, Menlo Park, CA, USA, 1997a.

E. Charniak. Statistical Techniques for Natural Language Parsing. *AI Magazine*, 18(4):33–43, 1997b.

L. Cherry. PARTS - A System for Assigning Word Classes to English Text. Technical Report Computing Science #81, Bell Laboratories, USA, 1980.

K. Church. A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. In *Proceedings of the* 2^{nd} *Conference on Applied Natural Language Processing*, Austin, Texas, USA, 1988.

M. Ciura and S. Deorowicz. How to squeeze a lexicon. *Software: Practice and Experience*, 31(11):1077–1099, 2001.

M. Collins. A New Statistical Parser Based on Bigram Lexical Dependencies. In *Proceedings of the* 34^{th} *Annual Meeting on Association for Computational Linguistics*, Santa Cruz, CA, USA, 1996.

H. Cunningham. *Software Architecture for Language Engineering*. PhD thesis, University of Sheffield, Sheffield, UK, 2000.

D. Cutting, J. Kupiec, J. Pealersen, and P. Sibun. A Practical Part-of-Speech Tagger. In *Proceedings of the* 3^{rd} *Conference on Applied Natural Language Processing*, Trento, Italy, 1992.

W. Daelemans, J. Zavrel, P. Berck, and S. Gillis. MBT: a Memory-Based Part of Speech Tagger-Generator. In *Proceedings of the* 4^{th} *Workshop on Very Large Corpora*, Copenhagen, Denmark, 1996.

R. Dale. Symbolic Approaches to Natural Langugae Processing. In R. Dale, H. Moisl, and H. Somers, editors, *Handbook of Natural Language Processing*. Marcel Dekker, New York, 2000.

T. G. Dietterich. Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. *Neural Computation*, 10(7):1895–1924, 1998.

S. Džeroski, T. Erjavec, and J. Zavrel. Morphosyntactic Tagging of Slovene: Evaluating Taggers and Tagsets. In *Proceedings of the* 2^{nd} *International Conference on Language Resources and Evaluation*, Athens, Greece, 2000.

EAGLES, Expert Advisory Group for Language Engineering Standards. Recommendations for the syntactic annotation of corpora. Technical report, 1996. http://www.ilc.cnr.it/EAGLES96/home.html. Accessed 15.02.2006.

J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.

D. Elworthy. Tagset Design and Inflected Languages. In *European Chapter of the ACL SIGDAT workshop "From Texts to Tags: Issues in Multilingual Language Analysis"*, Dublin, Ireland, 1995.

N. Ezeiza, I. Alegria, J. Arriola, R. Urizar, and I. Aduriz. Combining stochastic and rule-based methods for disambiguation in agglutinative languages. In *COLING-ACL '98:* 36^{th} *Annual Meeting of the Association for Computational Linguistics and* 17^{th} *International Conference on Computational Linguistics*, Montreal, Quebec, Canada, 1998.

R. Gaizauskas, M. Hepple, and C. Huyck. A Scheme for Comparative Evaluation of Diverse Parsing Systems. In *Proceedings of the* 1^{st} *International Conference on Language Resources and Evaluation*, Granada, Spain, 1998.

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

G. Grefenstette and P. Tapanainen. What is a word, What is a sentence? Problems of Tokenization. In *Proceedings of the* 3^{rd} *International Conference on Computational Lexicography*, Budapest, Hungary, 1994.

A. Grishman, A. Macleod, and A. Sterling. Evaluating parsing strategies using standardized parse files. In *Proceedings of the* 3^{rd} *Conference on Applied Natural Language Processing*, Trento, Italy, 1992.

K. Hagen, J. Johannessen, and A. Nøklestad. A Constraint-Based Tagger for Norwegian. In C.-E. Lindberg and S.-N. Lund, editors, 17^{th} *Scandinavian Conference of Linguistics. Odense Working Papers in Language and Communication*, volume 19, pages 31–48. Odense, Denmark, 2000.

J. Hajič. Morphological tagging: Data vs. Dictionaries. In *Proceedings of the* 1^{st} *Conference on North American Chapter of the ACL*, Seattle, WA, USA, 2000.

J. Hajič. Building a Syntactically Annotated Corpus: The Prague Dependency Tree-bank. In *Issues of Valency and Meaning*. Karolinum, Prague, 1998.

J. Hajič and B. Hladká. Czech Language Processing – PoS Tagging. In *Proceedings of the* 1^{st} *International Conference on Language Resources and Evaluation*, Granada, Spain, 1998.

J. Hajič, P. Krbec, K. Oliva, P. Květoň, and V. Petkevič. Serial Combination of Rules and Statistics: A Case Study in Czech Tagging. In *Proceedings of the* 39^{th} *Association of Computational Linguistics Conference*, Toulouse, France, 2001.

D. Hardt. Comma checking in Danish. In *Proceedings of the Corpus Linguistics Conference*, Lancaster, United Kingdom, 2001.

S. Helgadóttir. Testing Data-Driven Learning Algorithms for PoS Tagging of Icelandic. In H. Holmboe, editor, *Nordisk Sprogteknologi 2004*. Museum Tusculanums Forlag, Copenhagen, 2004.

E. Hinrichs and J. Trushkina. Getting a Grip on Morphological Disambiguation. In *Proceedings of KONVENS 2002, 6. Konferenz zur Verarbeitung natürlicher Sprache*, Saarbrücken, Germany, 2002.

D. Jurafsky and J. Martin. *Speech and Language Processing*. Prentice Hall, NJ, USA, 2000.

R. Kaplan. Formal Issues in Lexical-Functional Grammar. In C.-R. Huang and K.-J. Chen, editors, *Proceedings of ROCLING II*, 1989.

F. Karlsson. Constraint Grammar as a Framework for Parsing Running Text. In H. Karlgren, editor, *Papers presented to the* 13^{th} *International Conference on Computational Linguistics*, Helsinki, Finland, 1990.

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

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

G. Klein. JFlex User's Manual. Technical Report 1.4.1, July 2005. URL `http://jflex.de/manual.html`.

S. Klein and R. Simmons. A computational approach to grammatical coding of English words. *Journal of the ACM*, 10:334–347, 1963.

O. Knutsson, J. Bigert, and V. Kann. A Robust Shallow Parser for Swedish. In *Proceedings of NoDaLiDa 2003*, Reykjavik, Iceland, 2003.

D. Kokkinakis and S. Johansson-Kokkinakis. A Cascaded Finite-State Parser for Syntactic Analysis of Swedish. In *Proceedings of the* 9^{th} *Conference on European Chapter of the ACL (EACL)*, Bergen, Norwsay, 1999.

K. Koskenniemi. *Two-level morphology: A general computational model for word-form recognition and production*. PhD thesis, University of Helsinki, Helsinki, Finnland, 1983. Publication No. 11.

K. Koskenniemi. A General Computational Model for Word-Form Recognition and Production. In *Proceedings of COLING-84*, Stanford, CA, USA, 1984.

K. Koskenniemi. Finite-state parsing and disambiguation. In H. Karlgren, editor, *Papers presented to the* 13^{th} *International Conference on Computational Linguistics*, Helsinki, Finnland, 1990.

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

B. Kouchnir. Knowledge-Poor Grammatical Function Assignment for German. Technical report, Universität Tübingen, Tübingen, Germany, 2004. Manuscript. Seminar für Sprachwissenschaft.

S. Krauwer. ELSENET and ELRA: A common past and a common future, 1998. `www.elda.org/blark/fichiers/elsnet&elra.doc`. Accessed 01.03.2007.

S. Krauwer, B. Maegaard, K. Choukri, and L.-D. Jørgensen. *BLARK for Arabic*. Center for Sprogteknologi, 2004. NEMLAR project, `http://www.nemlar.org/Publications/BLARK-final_190906.pdf`. Accessed 01.03.2007.

A. Kuba, A. Hócza, and J. Csirik. POS Tagging of Hungarian with Combined Statistical and Rule-Based Methods. In *Proceedings of the International Conference on Text, Speech and Dialogue*, Brno, Czech Republic, 2004.

S. Kübler and T. Heike. Towards a Dependency-based Evaluation for Partial Parsing. In *Proceedings of Beyond PARSEVAL – Towards Improved Evaluation Measures for Parsing Systems (LREC 2002 Workshop)*, Las Palmas, Spain, 2002.

J. Kupiec. Robust part-of-speech tagging using a hidden Markov model. *Computer Speech and Language*, 6:225–242, 1992.

W. Lezius. Morphy – German Morphology, Part-of-Speech Tagging and Applications. In *Proceedings of the* 9^{th} *EURALEX International Congress*, Stuttgart, Germany, 2000.

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

D. Lin. A dependency-based method for evaluating broad-coverage parsers. *Natural Language Engineering*, 4(2):97–114, 1998.

D. Magerman and M. Marcus. Pearl: A Probabilistic Chart Parser. In *Proceedings of the European ACL Conference*, Berlin, Germany, 1991.

C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, fifth edition, 2002.

M. Marcus, B. Santorini, and M. Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2): 313–330, 1993.

M. Marcus, G. Kim, M. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. The Penn Treebank: Annotating Predicate Argument Structure. In *Proceedings of the ARPA Human Language Technology Workshop*, Plainsboro, NJ, USA, 1994.

B. Megyesi. *Data-driven Syntactic Analysis: Methods and Applications for Swedish*. PhD thesis, KTH, Stockholm, Sweden, 2002.

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

Menntamálaráðuneytið. *Samspil tungu og tækni – Afrakstur tungutækniverkefnis menntamálaráðuneytisins*. Menntamálaráðuneytið, Reykjavik, Iceland, 2004.

B. Merialdo. Tagging English Text with a Probabilistic Method. *Computational Linguistics*, 20(2):155–171, 1994.

A. Mikheev. Automatic Rule Induction for Unknown Word Guessing. *Computational Linguistics*, 21(4):543–565, 1997.

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

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

P. Nakov, Y. Bonev, G. Angelova, E. Cius, and W. Hahn. Guessing Morphological Classes of Unknown German Nouns. In *Proceedings of Recent Advances in Natural Language Processing*, Borovets, Bulgaria, 2003.

G. Ngai and R. Florian. Transformation-Based Learning in the Fast Lane. In *Proceedings of the* 2^{nd} *Conference of the North American Chapter of the ACL*, Pittsburgh, PA, USA, 2001.

J. Nivre. 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, 2002.

J. Nivre and J. Hall. Maltparser: A Language-Independent System for Data-Driven Dependency Parsing. In *Proceedings of the Fourth Workshop on Treebanks and Linguistic Theories*, Barcelona, Spain, 2005.

J. Nivre and M. Scholz. Deterministic Dependency Parsing of English Text. In *Proceedings of COLING 2004*, Geneva, Switzerland, 2004.

R. Ólafsson. *Tungutækni: Skýrsla starfshóps.* The Icelandic Ministry of Education, Science and Culture, Reykjavik, Iceland, 1999. `http://www.tungutaekni.is/news/Skyrsla.pdf`. Accessed 01.10.2004.

C. Oravecz and P. Dienes. Efficient Stochastic Part-of-Speech Tagging for Hungarian. In *Proceedings of the* 3^{rd} *International Conference on Language Resources and Evaluation*, Las Palmas, Spain, 2002.

D. Palmer. Tokenisation and Sentence Segmentation. In R. Dale, H. Moisl, and H. Somers, editors, *Handbook of Natural Language Processing.* Marcel Dekker, New York, 2000.

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

C. Pollard and I. Sag. *Head-driven Phrase Structure Grammar.* Center for the Study of Language and Information (CSLI) Lecture Notes. Stanford University Press and University of Chicago Press, USA, 1994.

H. Þráinsson. Icelandic. In E. König and J. Auwera, editors, *The Germanic Languages*, pages 142–189. Routledge, London, 1994.

A. Ratnaparkhi. A Maximum Entropy Model for Part-Of-Speech Tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Philadelphia, PA, USA, 1996.

E. Roche and Y. Schabes. Deterministic Part-of-Speech Tagging with Finite-State Transducers. In E. Roche and Y. Schabes, editors, *Finite-State Language Processing*. MIT Press, 1997.

E. Rögnvaldsson. The Corpus of Spoken Icelandic and its Morphosyntactic Annotation. In P. Henrichsen, editor, *Proceedings of the Seminar on Treebanks and Spoken Language*, Copenhagen Studies in Language, Stockholm, Sweden, 2005.

C. Samuelsson. Morphological tagging based entirely on Bayesian inference. In R. Eklund, editor, 9^{th} *Scandinavian Conference on Computational Linguistics*, Stockholm, Sweden, 1994.

C. Samuelsson and A. Voutilainen. Comparing a Linguistic and a Stochastic tagger. In *Proceedings of the* 8^{th} *Conference on European Chapter of the ACL (EACL)*, Madrid, Spain, 1997.

M. Schiehlen. A cascaded finite-state parser for German. In *Proceedings of the* 10^{th} *Conference on European Chapter of the ACL (EACL)*, Budapest, Hungary, 2003.

H. Schmid. Improvements in Part-of-Speech Tagging with an Application to German. In *European Chapter of the ACL SIGDAT workshop*, Dublin, Ireland, 1995.

S. Sekine and M. Collins. The Evalb Software. http://nlp.cs.nyu.edu/evalb/. Accessed November 2006, 1997.

M. Silberztein. The Lexical Analysis of Natural Languages. In E. Roche and Y. Schabes, editors, *Finite-State Language Processing*. MIT Press, 1997.

J. Sjöbergh. Combining POS-taggers for improved accuracy on Swedish text. In *Proceedings of NoDaLiDa 2003*, Reykjavik, Iceland, 2003.

Á. Svavarsdóttir. Icelandic. In U. Ammon and H. Harmann, editors, *Wieser Enzyklopädie der Sprachen Westeuropas (WSEW)*. Wieser Verlag, Klagenfurt, 2005.

P. Tapanainen and A. Voutilainen. Tagging accurately - Don't guess if you know. In *Proceedings of the* 4^{th} *ACL Conference on Applied Natural Language Processing*, Stuttgart, Germany, 1994.

H. Uszkoreit and A. Zaenen. Grammar Formalisms. In R. Cole, J. Mariani, H. Uszkoreit, A. Zaenen, and V. Zue, editors, *Survey of the State of the Art in Human Language Technology*. Cambridge University Press, 1995.

H. van Halteren. Performance of Taggers. In H. van Halteren, editor, *Syntactic Wordclass Tagging*. Kluwer Academic Publishers, 1999.

H. van Halteren, J. Zavrel, and W. Daelemans. Improving Data Driven Wordclass Tagging by System Combination. In *COLING-ACL '98:* 36^{th} *Annual Meeting of the Association for Computational Linguistics and* 17^{th} *International Conference on Computational Linguistics*, Montreal, Quebec, Canada, 1998.

H. van Halteren, J. Zavrel, and W. Daelemans. Improving Accuracy in Wordclass Tagging through Combination of Machine Learning Systems. *Computational Linguistics*, 27(2):199–230, 2001.

A. Voutilainen. A syntax-based part-of-speech analyzer. In *Proceedings of the* 7^{th} *Conference on European Chapter of the ACL (EACL)*, Dublin, Ireland, 1995.

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

D. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

D. H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208, 1967.

Appendix A

Abbreviations

Abbreviation	Explanation
AdvP	Adverb phrase
AP	Adjective phrase
BLARK	Basic Language Resource Kit
CFG	Context-free grammar
CG	Constraint Grammar
CYK	Cocke-Younger-Kasami
DDM	Data-driven method
DDT	Data-driven tagger
DFA	Deterministic finite-state automaton
EngCG	English Constraint Grammar
fnTBL	A tagger based on transformation-based error-driven learning
GATE	General Architecture for Text Engineering
GDC	Grammar definition corpus
GF	Grammatical function
HPSG	Head-Driven Phrase-Structure Grammar
IFD	Icelandic Frequency Dictionary
IL	Institute of Lexicography at the University of Iceland
IT	Information Technology
ITE	Icelandic tagging experiment
LRBM	Linguistic rule-based method
LRBT	Linguistic rule-based tagger
LFG	Lexical Functional Grammar
LOB	Lancaster-Oslo/Bergen
LT	Language Technology
MBT	A tagger based on memory-based learning
MESC	Icelandic Ministry of Education, Science and Culture
MXPOST	A tagger based on maximum entropy
MWE	Multiword expression
NL	Natural Language
NLP	Natural Language Processing
NP	Noun phrase
PCFG	Probabilistic Context-free grammar
POS	Part-of-speech
PP	Preposition phrase
TBL	Transformation-based learning
TnT	Trigrams and Tags – a tagger based on trigrams
UCREL	University Center for Computer Research on Language
VP	Verb phrase
XFST	Xerox finite-state tool
XRC	Xerox Research Centre

Table A.1: Abbreviations used in text

Appendix B

The Icelandic Tagset

Char#	Category/Feature	Symbol – semantics
1	Word class	**n**–noun
2	Gender	**k**–masculine, **v**–feminine, **h**–neuter, **x**–unspecified
3	Number	**e**–singular, **f**–plural
4	Case	**n**–nominative, **o**–accusative, **þ**–dative, **e**–genitive
5	Article	**g**–with suffixed definite article
6	Proper noun	**m**–person name, **ö**–place name, **s**–other proper name
1	Word class	**l**–adjective
2	Gender	**k**–masculine, **v**–feminine, **h**–neuter
3	Number	**e**–singular, **f**–plural
4	Case	**n**–nominative, **o**–accusative, **þ**–dative, **e**–genitive
5	Declension	**s**–strong declension, **v**–weak declension, **o**–indeclineable
6	Degree	**f**–positive, **m**–comparative, **e**–superlative
1	Word class	**f**–pronoun
2	Subcategory	**a**–demonstrative, **b**–reflexive, **e**–possessive, **o**–indefinite, **p**–personal, **s**–interrogative, **t**–relative
3	Gender/Person	**k**–masculine, **v**–feminine, **h**–neuter/**1**–1^{st} person, **2**–2^{nd} person
4	Number	**e**–singular, **f**–plural
5	Case	**n**–nominative, **o**–accusative, **þ**–dative, **e**–genitive
1	Word class	**g**–article
2	Gender	**k**–masculine, **v**–feminine, **h**–neuter
3	Number	**e**–singular, **f**–plural
4	Case	**n**–nominative, **o**–accusative, **þ**–dative, **e**–genitive
1	Word class	**t**–numeral
2	Category	**f**–cardinal, **o**–numeric constant, **p**–percentage
3	Gender	**k**–masculine, **v**–feminine, **h**–neuter
4	Number	**e**–singular, **f**–plural
5	Case	**n**–nominative, **o**–accusative, **þ**–dative, **e**–genitive
1	Word class	**s**–verb (except for past participle)
2	Mood	**n**–infinitive, **b**–imperative, **f**–indicative, **v**–subjunctive, **s**–supine, **l**–persent participle
3	Voice	**g**–active, **m**–middle
4	Person	**1**–1^{st} person, **2**–2^{nd} person, **3**–3^{rd} person,
5	Number	**e**–singular, **f**–plural
6	Tense	**n**–present, **þ**–past
1	Word class	**s**–verb (past participle)
2	Mood	**þ**–past participle
3	Voice	**g**–active, **m**–middle
4	Gender	**k**–masculine, **v**–feminine, **h**–neuter
5	Number	**e**–singular, **f**–plural
6	Case	**n**–nominative, **o**–accusative, **þ**–dative, **e**–genitive
1	Word class	**a**–adverb and preposition
2	Category	**a**–does not govern case, **u**–exclamation, **o**–governs accusative, **þ**–governs dative, **e**–governs genitive
3	Degree	**m**–comparative, **e**–superlative
1	Word class	**c**–conjunction
2	Category	**n**–sign of infinitive, **t**–relativizer,
1	Word class	**e**–foreign word
1	Word class	**x**–unanalyzed word

Table B.1: The Icelandic tagset

Appendix C

IceParser – The Finite-State Transducers

IceParser is implemented in Java and the lexical analyser generator tool JFlex (http://jflex.de/). It consists of two main components, a phrase structure module and a syntactic functions module. The input to the parser is POS-tagged sentences. The tags are assumed to be part of the tagset used in the *IFD* corpus, i.e. the tagset used by *IceTagger*. Figure C.1 shows the architecture of *IceParser*.

C.1 The phrase structure module

This module consists of 13 transducers. Ten of these mark phrase structure, two add case information to phrases, and one takes care of "clean up". Case information is added to the phrase types *APs* and *NPs* in order to facilitate the marking of a sequence of adjective and noun phrases.

All the phrase structure transducers include the file *phraseDef.txt*, which defines a number of base patterns used by the transducers. In the examples which follow, we do not explicitly show this include statement.

Let us now discuss each of these transducers, in the (approximate) order in which they are executed.

1. **Phrase_MWEP1**: This is a simple transducer which puts the markers *[MWE_PP ... MWE_PP]* around multiword expressions (MWEs) consisting of specific <preposition, adverb> pairs. The <preposition, adverb> pairs consists of the preposition "*fyrir*" followed by any of the

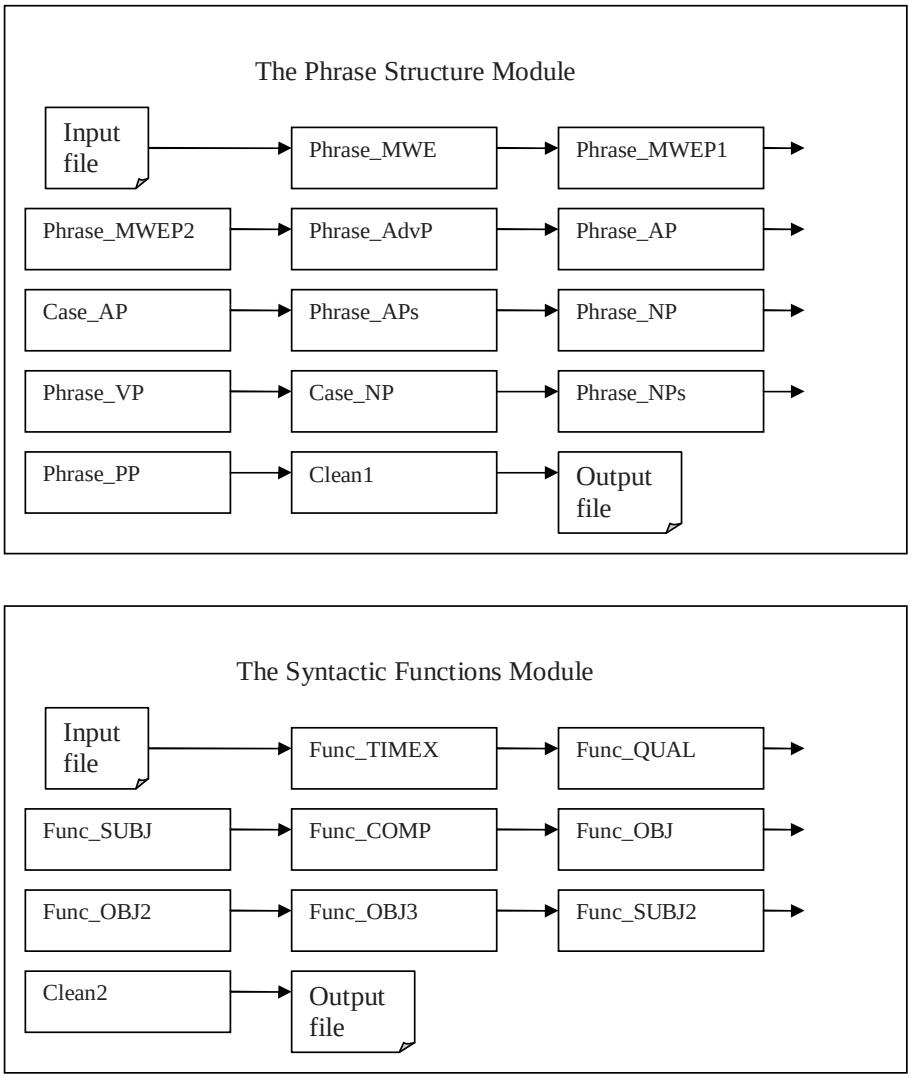

Figure C.1: The architecture of *IceParser*

adverbs "*aftan*", "*austan*", "*framan*", "*neðan*", "*norðan*", "*ofan*", "*sunnan*", "*utan*", "*vestan*" (for example, "*fyrir framan*" = in front).

The JFlex code for this transducer, despite being simple, is a good illustrative example of the format of each of the finite-state transducers:

```
%public
%class Phrase_MWEP1
%standalone
%line
%unicode

%{
  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}      { System.out.print(Open+yytext()+Close);}
```

The options at the top of the file (prefixed with the percentage sign) instruct the JFLex compiler to generate a public class "Phrase_MWEP1", with a main function (because of the %standalone directive) that expects a name of an input file on the command line. Furthermore, the generated program should use the full 16-bit Unicode input character set, and line counting should be turned on.

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

Two regular definitions[1], *AdverbPart* and *PrepPart*, define the adverb part and the preposition part of the <preposition, adverb> pair, re-

[1]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.

spectively. For example, the adverb part consists of one or more white spaces ({WS}+) followed by an *AdverbTag*. The *AdverbTag* is a name defined in the file *phraseDef.txt* (*PrepPart* is defined similarly):

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

i.e. the letters *aa* optionally followed by the letters *m* or *e* (see the description of the Icelandic tagset in Appendix B) and postfixed with one or more white spaces.

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

2. **Phrase_MWEP2**: This transducer marks MWEs consisting of specific <adverb, preposition> pairs by putting the markers *[MWE_PP ... MWE_PP]* around the pairs. Examples of <adverb, preposition> pairs are: "*út um*" (out of), "*upp í*" (up to), "*ofan á*", (onto) "*niður í*", (down to) and "*upp fyrir*" (up to).

   Note that the transducers **Phrase_MWEP1** and **Phrase_MWEP2** can not be combined into one transducer because of sentence parts like "*upp fyrir aftan*", in which "*fyrir aftan*" (behind) should be recognised as a unit, but not "*upp fyrir*".

3. **Phrase_MWE**: This transducer marks MWEs consisting of common bigrams and trigrams. It puts the markers *[MWE_AdvP ... MWE_AdvP]*, *[MWE_AP ... MWE_AP]* or *[MWE_CP ... MWE_CP]* around MWEs that function as an adverb, an adjective, or a conjunction, respectively.

   The code for this transducer includes a list of about 70 "adverb" MWEs, 15 "adjective" MWEs, and 20 "conjunction" MWEs. For example, the MWE "*hins vegar*" (on the other hand) functions as an adverb and is marked as *[MWE_AdvP hins fakee vegar nkee MWE_AdvP]* by this transducer; "*þess háttar*" (such kind) functions as an adjective and is marked as *[MWE_AP þess fakee háttar nkee MWE_AP]*, and "*af því*

að" (because) functions as a conjunction and is marked as *[MWE_CP af aþ því fpheþ að c MWE_CP]*.

The complete list of multiword expressions recognised by *IceParser* can be found in Appendix C.4.

4. **Phrase_AdvP**: The main function of this transducer is to mark AdvPs, consisting of a single adverb, by putting the markers *[AdvP ... AdvP]* around them. An adverb in the input text is recognised using the patterns:

```
WordChar = [^\r\n\t\f ]
Word = {WordChar}+
WordSpaces = {WS}*{Word}{WS}+
Adverb = {WordSpaces}{AdverbTag}
```

According to these patterns, an adverb is sequence of word characters (all possible characters except white spaces) followed by at least one space followed by an adverb tag.

Additionally, this transducer marks coordinating conjunction phrases, *[CP ... CP]*, subordinating conjunction phrases, *[SCP ... SCP]*, and interjection phrases, *[InjP ... InjP]*, consisting of a single conjunction and interjection, respectively.

5. **Phrase_AP**: This transducer marks adjective phrases, *[AP ... AP]*, which consist of a single adjective optionally preceded by an AdvP. It uses the following patterns:

```
Gender = [kvhx]     /* k=masc., v=fem., h=neut., x=unspec */
Number = [ef]       /* e=singular, f=plural */
ObliqueCase = [oþe] /* o=acc., þ=dat., e=gen. */
Case = n | {ObliqueCase}    /* n=nom. */
Degree = [fme]              /* f=pos., m=comp., e=superl. */
Declension = [osv]  /* o=no, s=strong, v=weak */

OpenAdvP = "[AdvP"
CloseAdvP = "AdvP]"
```

```
AdjectiveTag = l{Gender}{Number}{Case}{Declension}{Degree}
Adjective = {WordSpaces}{AdjectiveTag}
AdverbPhrase = {OpenAdvP}~"aa"{WS}+{CloseAdvP}
AdjectivePhrase = {AdverbPhrase}?{Adjective}
```

In JFlex, the regular expression $\sim a$ matches everything up to (and including) the first occurrence of a text matched by a. Thus, an AdvP, to be included in an AP, consist of a bracket and a label denoting the start of an adverb phrase ({OpenAdvP}) followed by everything up to the tag *aa*, followed by at least one whitespace and a label and a bracket denoting the end of the AdvP ({CloseAdvP}). The reason for explicitly specifying the *aa* tag is that adverbs in comparative or superlative (the tags *aam* or *aae*) are never a part of an AP.

To illustrate, the AP "*mjög bjart*" (very bright) is annotated as *[AP [AdvP mjög aa AdvP] bjart lhensf AP]*.

6. **Case_AP**: This transducer adds case information to adjective phrases. As previously mentioned, this is performed in order to facilitate marking of a sequence of adjective phrases.

   The transducer searches for AP patterns, *[AP ... AP]*, extracts the case information from the last tag of the last word (the head) of the phrase and appends the case character to the starting label of the phrase.

   For example, lets assume the phrase *[AP andstyggilegur lkensf AP]* (disgusting) is part of the input into this transducer. Then, when this AP is recognised, a specific function is called, with the tag *lkensf* as an argument, which extracts the case character (*n*) from the tag and returns it. The action, associated with the recognised pattern, appends the returned case to the starting label, resulting in the string *[APn andstyggilegur lkensf AP]*, denoting an nominative AP.

7. **Phrase_APs**: The purpose of this transducer is to group together a sequence of adjective phrases (APs), which form a single constituent. The APs are adjacent to each other, optionally separated by a comma or a conjunction. Furthermore, all adjacent APs that form a single constituent do agree in case. Thus, in addition to adjacent conjunction phrases or a comma, this transducer uses the case information added by the previous transducer to facilitate the grouping:

```
APConjNom = ({WS}+({ConjPhraseOrComma}{WS}+)?{APNom})+
APConjAcc = ({WS}+({ConjPhraseOrComma}{WS}+)?{APAcc})+
APConjDat = ({WS}+({ConjPhraseOrComma}{WS}+)?{APDat})+
APConjGen = ({WS}+({ConjPhraseOrComma}{WS}+)?{APGen})+

APSeq = {APNom}{APConjNom} | {APAcc}{APConjAcc} |
        {APDat}{APConjDat} | {APGen}{APConjGen}
```

According to the patterns above, a sequence of APs, *APSeq*, consists of at least two adjective phrase, marked by one of nominative, accusative, dative, or genitive case, optionally separated by a conjunction phrase or a comma (*ConjPhraseOrComma*). The definition of an AP marked by a specific case is, for example:

```
OpenAP = "[AP"
CloseAP = "AP]"
APNom = {OpenAP}n~{CloseAP}
```

where, the letter *n* denotes the nominative case.

The pattern *ConjPhraseOrComma* is defined as follows:

```
Comma = ","{WS}+","
ConjPhrase = {OpenCP}~{CloseCP}
ConjPhraseOrComma = {ConjPhrase} | {Comma}
```

The pattern consists either of a coordinating conjunction phrase or the single lexeme comma.

To illustrate, assuming the input into this transducer consists of annotated phrases like:

[APn gul lhfnsf AP] , , [APn rauð lhfnsf AP] (yellow, red) and *[APn marðir lkfnsf AP] [CP og c CP] [APn særðir lkfnsf AP]* (bruised and wounded)

This transducer will then produce the output:

[APs [APn gul lhfnsf AP] , , [APn rauð lhfnsf AP] APs] and *[APs [APn marðir lkfnsf AP] [CP og c CP] [APn særðir lkfnsf AP] APs].*

8. **Phrase_NP**: This transducer marks noun phrases. It is the most complicated of all the transducers, because noun phrases can be formed in various ways (see section 6.2.1.7).

   The implementation uses various patterns for noun phrases depending on the first word/tag of the phrase. A noun phrase (NP) can, for example, start with interrogative pronouns "*hvað*" (what), "*hver*" (who) or "*hvor*" (which), a reflexive pronoun (like "*sig*" (himself)), a demonstrative pronoun, an indefinite pronoun, a personal pronoun, an article, etc.

   The implementation uses the main word order for an Icelandic NP, i.e. the order described in section 6.2.1.7. An example of a (complex) NP is "*allir þessir þrír stóru strákar*" (all these three big boys), which consists of an indefinite pronoun, demonstrative pronoun, numeral, adjective, and a noun, respectively.

   The main patterns used are (without showing all the details):

```
Hvada = {WS}*[Hh]vaða{WS}+{InterPronounTag}
Hvad = {WS}*[Hh]v(að|((er|or)[a-z]*)){WS}+{InterPronounTag}

AdjectivePhrase =   {OpenAP}~{CloseAP}
AdjectivePhrases =  {WS}*({OpenAPs}~{CloseAPs}|
                    {AdjectivePhrase} |{MWE_AP})

NounProperPoss =    {ReflexivePronoun}?({Noun}|{ProperNoun})
                    ({PossPronoun}|{Numeral})?
ReflNP = {ReflexivePronoun}({Noun}|{PersPronoun})?
HvadaNP = {Hvada}{Numeral}?{AdjectivePhrases}?{NounProperPoss}
NumNP = {Numeral}({AdjectivePhrases}?{NounProperPoss})?
ArticleNP = {Article}{Numeral}?{AdjectivePhrases}?{Noun}
PersNP = {PersPronoun}{ReflexivePronoun}?
PossNP = {PossPronoun}({AdjectivePhrases}?{Noun})?
DemonNP =   {DemonPronoun}({IndefPronoun} |
            {Numeral}?{AdjectivePhrases}?{NounProperPoss}?)
IndefNP =   {IndefPronoun}+({Article}|{DemonPronoun})?
            {Numeral}?{AdjectivePhrases}?{NounProperPoss}?
AdjAP = {AdjectivePhrases}{Numeral}?{NounProperPoss}
ProperNounNP = {Title}?{ProperNoun}+
```

```
                ({ReflexivePronoun}|{PossPronoun})?
NounNP = {Noun}({ReflexivePronoun}|{Numeral}|{DemonPronoun}?
                 {IndefPronoun}|{PossPronoun})?

NounPhrase = {Hvad} | {HvadaNP} | {ReflNP} | {ArticleNP} |
        {DemonNP} | {IndefNP} | {PersNP} | {PossNP} |
        {NumNP} | {AdjAP} | {NounNP} | {ProperNounNP}
```

The {NounPhrase} is the main pattern which uses a number of sub-patterns for the various types of an NP. This transducer produces the following annotation for the complex NP above: *[NP Allir fokfn þessir fakfn þrír tfkfn [AP stóru lkfnvf AP] strákar nkfn NP].*

9. **Phrase_VP**: Annotation of verb phrases is carried out by this transducer. As discussed in section 6.2.1.9, six categories are used for verb phrases: *[VP ... VP]*, *[VPi ... VPi]*, *[VPb ... VPb]*, *[VPs ... VPs]*, *[VPp ... VPp]* and *[VPg ... VPg]*, denoting a finite VP, an infinitive VP, a VP consisting (mainly) of the verb "be", a supine VP, a past participle VP and a present participle VP, respectively. The information about the type of a VP facilitates the annotation of syntactic functions (see section C.2).

   The search for the different kinds of verb phrases is straight-forward, because the tag associated with each verb includes, in most cases, the information needed. Consider, for example, the code needed to recognise a past participle verb phrase (VP):

```
%{
  String VPIOpen=" [VPp ";
  String VPIClose=" VPp] ";
%}
VerbPastPartTag = sþ{Voice}{Gender}{Number}{Case}
VerbPhrasePastPart = {WordSpaces}{VerbPastPartTag}
%%
{VerbPhrasePastPart} {System.out.print(VPPOpen+yytext()+VPPClose);
```

   The *þ* letter in the second character of the *VerbPastPartTag* pattern is an indication of a past participle verb. The patterns for present participle and supine VPs are implemented in a similar manner.

Patterns for recognising infinitive VPs are:

```
%{
  String VPPOpen=" [VPp ";
  String VPPClose=" VPp] ";
%}
InfinitiveTag = cn
Infinitive = {WordSpaces}{InfinitiveTag}
VerbInfinitiveTag = sn[mg]{WS}+
VerbInfinitive = {WordSpaces}{VerbInfinitiveTag}
VerbPhraseInf = {Infinitive}?{VerbInfinitive}{VerbSupine}*
%%
{VerbPhraseInf} {System.out.print(VPIOpen+yytext()+VPIClose);}
```

According to these patterns, an infinitive VP (*VerbPhraseInf*) consists of the (optional) infinitive (a word tagged as *cn*) followed by a verb tagged as an infinitive verb (*sng* or *snm*), followed by zero or more supine verbs. An example of output generated by this transducer for an infinitive VP is: *[VPi að cn laga sng VPi]* (to fix).

VPs consisting of the verb "be", and a few other verbs which demand nominative predicates, are found using patterns that simply search for all possible word forms for these verbs, which have a finite verb POS tag.

Finally, a finite verb phrase (not including "be" verbs) is recognised using the patterns:

```
%{
  String VPOpen=" [VP ";
  String VPClose=" VP] ";
%}
VerbFiniteTag = s[bfv]{Voice}{Person}{Number}{Tense}{WS}+
VerbFinite =  {WordSpaces}{VerbFiniteTag}
VerbPhrase =  {VerbFinite}
                (({WS}*{AdverbPhrase})?{VerbSupine}+)?
%%
{VerbPhrase} {System.out.print(VPOpen+yytext()+VPClose);}
```

A finite VP, thus, consists of a verb tagged with a finite verb POS tag, optionally followed by an adverb phrase and one or more supine verbs. Examples of output generated by this transducer for a finite VP are: *[VP sagði sfg1eþ VP]* (said) and *[VP hafði sfg3eþ [AdvP líklega aa AdvP] farið ssg VP]* (had probably gone).

10. **Case_NP**: This transducer adds case information to noun phrases. This is carried out in order to group a sequence of noun phrases (see the description of the next transducer). The transducer searches for NP patterns, *[NP ... NP]*, extracts the case information from the last tag of the last word (the head) of the phrase and appends the case character to the starting label of the phrase. Thus, the functionality of this transducer is similar to the one described for the *Case_AP* transducer (see above).

11. **Phrase_NPs**: The purpose of this transducer is to group together a sequence of noun phrases (NPs), which form a single constituent. The NPs are adjacent to each other, and optionally separated by a comma or a conjunction. Furthermore, all adjacent NPs that form a single constituent do agree in case. The functionality of this transducer differs from the **Phrase_APs** (described above) in that a genitive NP qualifier phrase can be part of the grouping. Thus, in addition to adjacent conjunction phrases or a comma, this transducer uses the case information added by the previous transducer to facilitate the grouping:

```
NPNomGenQual = {NPNom}{GenQualifier}*
NPAccGenQual = {NPAcc}{GenQualifier}*
NPDatGenQual = {NPDat}{GenQualifier}*

CommaNPNom = {Comma}{WS}+{NPNom}{WS}+
CommaNPAcc = {Comma}{WS}+({NPAcc}|{APAcc}){WS}+
CommaNPDat = {Comma}{WS}+({NPDat}|{APDat}){WS}+
CommaNPGen = {Comma}{WS}+{NPGen}{WS}+

NPConjNom = {CommaNPNom}*{ConjPhrase}{WS}+{NPNom}
NPConjAcc = {CommaNPAcc}*{ConjPhrase}{WS}+({NPAcc}|{APAcc})
NPConjDat = {CommaNPDat}*{ConjPhrase}{WS}+({NPDat}|{APDat})
NPConjGen = {CommaNPGen}*{ConjPhrase}{WS}+{NPGen}
```

```
NPSeq = {NPProperNom}{WS}+{NPNom}           |
    {NPProperAcc}{WS}+{NPAcc}           |
    {NPProperDat}{WS}+{NPDat}           |
    {NPProperGen}{WS}+{NPGen}           |

    {NPNomGenQual}{WS}+({NPProperNom}|{NPConjNom})  |
    {NPAccGenQual}{WS}+({NPProperAcc}|{NPConjAcc})  |
    {NPDatGenQual}{WS}+({NPProperDat}|{NPConjDat})  |
    {NPGen}{WS}+({NPProperGen}|{NPConjGen})

%%
{NPSeq} { System.out.print(NPOpen+yytext()+NPClose);}
```

All the details are not shown above, but, according to the patterns, a sequence of NPs, *NPSeq*, consists of a noun phrase marked by one of the four cases, and optionally followed by a genitive qualifier, followed by a sequence of noun phrases agreeing in case and separated by a comma and finally a conjunction phrase. The definition of a NP marked by a specific case is, for example:

```
OpenAP = "[NP"
CloseAP = "NP]"
NPNom = {OpenNP}n~{CloseNP}
```

where, as before, the letter *n* denotes the nominative case.

To illustrate, assuming the input into this transducer consists of substrings like:

- [NPn börn NP] [NPg hans NP] [CP og CP] [NPn niðjar NP] (children his and descendants)
- [NPd fiskum NP] , [NPd liðdýrum NP] [CP og CP] [NPd spendýrum NP] NPs] (fish, arthropods and mammals)

. . . the transducer will produce the output:

- [NPs [NPn börn NP] [NPg hans NP] [CP og CP] [NPn niðjar NP] NPs]

- [NPs [NPd fiskum NP] , [NPd liðdýrum NP] [CP og CP] [NPd spendýrum NP] NPs].

12. **Phrase_PP**: PPs are annotated by this transducer. A PP consists of either a preposition followed by an infinitive VP, or a preposition followed by a single NP or a sequence of NPs. Single accusative or dative NPs can, additionally, be followed by a genitive qualifier NP, denoting possession.

    The NPs can optionally be preceded by an adverb phrase or a multiword expression, functioning as an adverb phrase (*[MWE_PP ... MWE_PP]*. Furthermore, one special case need to be accounted for, i.e. the preposition "*vegna*" can appear after the noun phrase(s).

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

Vegna = {WS}*vegna{WS}+

NounPhrase = {OpenNP}~{CloseNP}
NPs = {OpenNPs}~{CloseNPs}
AdverbPrepPhrase = {MWE_PP}

NPGenSpec = {OpenNP}g~f(p|s)hee{WS}+{CloseNP}
AdjectivePhrase = {OpenAP}~{CloseAP}

AdverbPhrase = {OpenAdvP}~{CloseAdvP} | {MWE_AdvP}
VerbPhraseInf = {OpenVPi}~{CloseVPi}

GenQualifier = {WS}*({NPGen} | {NPsGen})

OneNP = ({NounPhrase}|{AdjectivePhrase}){GenQualifier}?
SeqNP = {NPs}{GenQualifier}?

PrepPhraseRest1 = {AdverbPhrase}{WS}+({OneNP}|{SeqNP}) |
                    {OneNP} | {SeqNP}
PrepPhraseRest2 = {VerbPhraseInf}
```

```
PrepPhraseRest3 = {GenQualifier}{WS}+({OneNP}|{SeqNP})

PrepPhraseAccDat = {PrepositionAccDat}({PrepPhraseRest1}|
                    {PrepPhraseRest2}|{PrepPhraseRest3})?
PrepPhraseGen = {PrepositionGen}
                ({PrepPhraseRest1}|{PrepPhraseRest2})?
PrepPhraseMWE = {AdverbPrepPhrase}{WS}+({OneNP}|{SeqNP})
PrepPhraseSpecial = {NPGenSpec}{Vegna}{PrepTagGen}

%%
{PrepPhraseSpecial} {System.out.print(Open+yytext()+Close);}
{PrepPhraseAccDat}  {System.out.print(Open+yytext()+Close);}
{PrepPhraseGen}     {System.out.print(Open+yytext()+Close);}
{PrepPhraseMWE}     {System.out.print(Open+yytext()+Close);}
```

Not all of the details are shown above, but, to summarise, this transducer marks PPs that i) start with an accusative or dative case preposition, and are followed by a) an adverb phrase/genetive qualifier phrase and one or more noun phrases or b) an infinitive phrase, or ii) start with a genitive case preposition, and are followed by a) an adverb phrase phrase and one or more noun phrases or b) an infinitive phrase, or iii) start with a multiword expression functioning as an preposition (MWE_PP), followed by one or more noun phrases, or iv) consist of a noun phrase (a pronoun) followed by preposition "*vegna*".

For example, given the input substrings below:

- í aþ [NPd [APd skuggsælu lheþsf AP] húsi nheþ NP] (in shadowy house)
- í aþ [NPd sögu nveþ NP] [NPg fjölskyldunnar nveeg NP] (in story family's)
- [MWE_PP úti aa við ao MWE_PP] [NPa sjóinn nkeog NP] (out by sea)
- í aþ [NPs [NPd haustmyrkri nheþ NP] [CP og c CP] [NPd vetrargnauði nheþ NP] NPs] (in autumn-darkness and winter-hiss)

. . . this transducer produces the corresponding output:

- [PP í aþ [NPd [APd skuggsælu lheþsf AP] húsi nheþ NP] PP]

- [PP í aþ [NPd sögu nveþ NP] [NPg fjölskyldunnar nveeg NP] PP]
- [PP [MWE_PP úti aa við ao MWE_PP] [NPa sjóinn nkeog NP] PP]
- [PP í aþ [NPs [NPd haustmyrkri nheþ NP] [CP og c CP] [NPd vetrargnauði nheþ NP] NPs] PP]

13. **Clean1**: The purpose of this transducer is mainly to correct three types of annotation errors that might have been made by the previous transducers in the sequence.

    5Hence, the **Clean1** transducer changes this to the correct string *[MWE_ CP til ae þess fphee að cn MWE_ CP]*.

    First, this transducer finds the occurrence of a nominative adjective phrase inside a dative noun phrase and converts it to two independent phrases. This pattern occurs when the adjective governs the case of the following noun, e.g. in the clause *"nátengdara sögu"* (closely-connected history). The previous transducers annotates this as *[NPd [APn nátengdara lhenvm AP] sögu nveþ NP]*, because the **Phrase_NP** transducer does not take the different cases into account, but, rather, assumes that the adjective modifies the noun. Since this is a common phenomenon, the **Clean1** transducer searches for this pattern and corrects the annotation (using string searches and replacements). For our example above, it returns the string *[APn nátengdara lhenvm AP] [NPd sögu nveþ NP]*.

    The **Phrase_NP** groups together two (or more) adjacent proper nouns. This is in most cases correct (e.g. *"Jón Jónsson")*, except when the second proper noun is, in fact, a genitive qualifier. This occurs, for example, in the clause *"Bandaríki Norður-Ameríku"* (United-States North-America's), which is annotated as *[NPg Bandaríki nhfn-ö Norður-Ameríku nvee-ö NP]* by the previous transducers. The second task of the **Clean1** transducer is to find these occurrences and correct the annotation (using string searches and replacements). In the above example, it returns the correct string *[NPn Bandaríki nhfn-ö NP] [NPg Norður-Ameríku nvee-ö NP]*

    Additionally, this transducer converts a sequence of adverb phrases into one phrases consisting of two or more adverbs. This can be carried out in a simple manner:

```
{AdvPSeq}   {
            String str = yytext();
            /* rm all adverb labels */
            str = str.replaceAll("\\[AdvP","");
            str = str.replaceAll("AdvP]","");
             /* Then add one instance */
            str = "[AdvP" + str + "AdvP] ";
            System.out.print(str);
        }
```

For an input string like *[AdvP bráðum aa AdvP] [AdvP aftur aa AdvP]*, this transducer outputs *[AdvP bráðum aa aftur aa AdvP]*.

C.2 The syntactic functions module

This module consists of 9 transducers, 8 of which mark functions and one takes care of "clean up". All the syntactic functions transducers include the files *phraseDef.txt* (describe in section C.1) and *funcDef.txt*, which defines a number of base functional patterns used by the transducers. In the examples which follow, we do not explicitly show these include statements. Moreover, we do not show POS tags in the examples.

Let us now discuss each of these transducers, in the order in which they are executed.

1. **Func_TIMEX**: This first transducer puts the markers *{*TIMEX ... *TIMEX}* around temporal expressions. The transducer mainly uses patterns that include numbers and names of months, e.g.:

```
%{
  String TempOpen = " {*TIMEX ";
  String TempClose=" *TIMEX} ";
%}

NumberTag = t[ao]{WS}+
NounNumeral = {WordSpaces}({NounTag}{WordSpaces}){NumberTag}
TimeAcc = {OpenNP}a{NounNumeral}{CloseNP}
TimeMonth = {OpenNP}a({WS}+{AdjPhrase}){Month}
```

```
            ({WordSpaces}{NumberTag})?{CloseNP}
OneNumber = {OpenNP}{WordSpaces}{NumberTag}{CloseNP}
Temporal = {TimeAcc} | {TimeMonth} | {OneNumber}

%%
{Temporal}  {System.out.print(TempOpen+yytext()+TempClose);}
```

According to these patterns, a Temporal is a noun phrase ({OpenNP} ...{CloseNP}), which includes numbers or names of the months (and can include an adjective phrase). Note that if the noun phrase does not only include a number ({OneNumber}), then only accusative noun phrases (the character *a*) are used.

For example, given the input substrings below:

- [NP átta NP] (eight)
- [NP árið 1982 NP] (year 1982)
- [NP [AP 3. AP] desember 1987 NP]

...the corresponding output is:

- {*TIMEX [NP átta NP] *TIMEX}
- {*TIMEX [NP árið 1982 NP] *TIMEX}
- {*TIMEX [NP [AP 3. AP] desember 1987 NP] *TIMEX}

2. **Func_QUAL**: This transducer puts the markers *{*QUAL ... *QUAL}* around genitive qualifier noun phrases. An exception are genitive noun phrases appearing inside a genitive PP. The patterns used in this transducer are the following:

```
%{
  String OpenQual="{*QUAL ";
  String CloseQual=" *QUAL}";
%}

PPSkip = {OpenPP}{PrepositionGen}{NPGen}
NPGenSeq = {OpenNPs}{WS}+{OpenNP}g~{CloseNPs}
NPQual = {NPGen}({WS}+{NPGen})* | {NPGenSeq}
```

```
%%
{PPSkip}    {System.out.print(yytext()); }
{NPQual}    { System.out.print(OpenQual+yytext()+CloseQual);}
```

Hence, either a qualifier is one or more single genitive noun phrases ({NPGen}) or a sequence of noun phrases, in which the first phrase is a genitive noun phrase ({NPGenSeq}).

To illustrate, given the input substrings below:

- [NPn systir NP] [NPg hennar NP] (sister hers)
- [NP [AP nýkjörinn AP] forseti NP] [NP lýðveldisins NP] (newly-elected president republic's)
- [PP á [NP tímum NP] [NPs [NP rútubíla NP] [CP og CP] [NP [AP mikilla AP] mannflutninga NP] NPs] PP]

. . . the corresponding output is:

- [NPn systir NP] {*QUAL [NPg hennar NP] *QUAL}
- [NP [AP nýkjörinn AP] forseti NP] {*QUAL [NP lýðveldisins NP] *QUAL}
- [PP á [NP tímum NP] {*QUAL [NPs [NP rútubíla NP] [CP og CP] [NP [AP mikilla AP] mannflutninga NP] NPs] *QUAL} PP]

3. **Func_SUBJ**: This transducer puts the markers *{*SUBJ . . . *SUBJ}*, *{*SUBJ> . . . *SUBJ>}* or *{*SUBJ< . . . *SUBJ<}* around subject noun phrases. Refer to section 6.2.2.2 for a description of how subjects should be annotated according to the annotation scheme.

   This transducer uses various patterns to recognise subjects, depending on whether the subject appears to the left of the finite verb phrase, to the right of the verb phrase, precedes a relativizer (i.e. a subordinating conjunction that links a relative clause to its head noun), etc. Here, we only discuss the main case, i.e. when the subject appears to the left of the finite verb phrase (other cases are implemented similarly):

```
NomSubject = {NPNom} | {NPsNom}
PPorQual = ({PP}{WS}+)+ | {FuncQualifier}{WS}+
```

```
VPorVPBe = {VP}|{VPBe}

SubjectVerb =
        ({FuncQualifier}{WS}+)?({NomSubject}|{NPNum})
         {WS}+{PPorQual}?{VPorVPBe}          |
         {DatSubject}{WS}+{PPorQual}?{VPDat}     |
         {AccSubject}{WS}+{PPorQual}?{VPAcc}
```

{FuncQualifier} matches a qualifier function (*{*QUAL ... *QUAL}*) and {NomSubject}, {AccSubject} and {DatSubject} match a single nominative NP, or a sequence of NPs in the nominative, accusative, or dative case, respectively. {NPNum} matches a NP consisting of a numeral and {PPorQual} matches a preposition or a qualifier function. {VPorVPBe} matches a finite verb phrase or a "be" verb phrase, and {VPDat} and {VPAcc} match verbs that demand oblique case subjects (this verb list is implemented as a regular expression).

The action associated with the pattern {SubjectVerb} is a bit more complicated than the actions we have seen for the other transducers presented hitherto. The reason is that the subject marking should only be put around a part of the recognised pattern. Neither a preposition phrase nor a verb phrase should be marked as a part of the subject. The action, thus, needs to find out where the subject, in the recognised substring, ends.

In our implementation, this is performed by simple string searches. If a PP is part of the recognised substring then the subject ends where the PP starts. Otherwise, the subject ends where the VP starts.

The {SubjectVerb} pattern matches subject-verb pairs in which the verb appears to the right of the subject. Therefore, the markers *{*SUBJ> ... *SUBJ>}* are used.

To illustrate, given the input substrings below ...

(a) [NPn ég NP] [VPb var VPb] (I was)

(b) [NP systir NP] {*QUAL [NP hennar NP] *QUAL} [VPb var VPb] (sister hers was)

... the corresponding output is:

(a) {*SUBJ> [NPn ég NP] *SUBJ>} [VPb var VPb] [APn lítill AP]

(b) {*SUBJ> [NP systir NP] {*QUAL [NP hennar NP] *QUAL} *SUBJ>} [VPb var VPb]

4. **Func_COMP**: This transducer puts the markers *{*COMP ... *COMP}*, *{*COMP> ... *COMP>}* or *{*COMP< ... *COMP<}* around verb complement phrases. Recall that complements are objects of verbs which demand a predicate nominative (mainly the "be" verb; see section 6.2.2.3). Here, we only discuss the main word order, i.e. Subject-Verb Phrase-Complement (variants of it are implemented similarly). Note that after the verb phrase (and before the complement) a qualifier, and adverb phrase or an accusative/dative case noun phrase might appear (see examples below).

   This transducer assumes that subjects have already been annotated by the previous transducer. The code for the main case is the following:

```
VPPastSeq =
    {VPPast}({WS}+{ConjPhraseOrComma}{WS}+{VPPast})*
Complement = {APNom}|{APsNom}|{NPNom}|{NPsNom}|{VPPastSeq}
SubjectVerbBe = {FuncSubject}{WS}+{VPBe}{WS}+

SubjVerbAdvPCompl =
    {SubjectVerbBe}{AdvP}{WS}+{Complement}
SubjVerbNPCompl =
    {SubjectVerbBe}({NPAcc}|{NPDat}){WS}+{Complement}
SubjVerbCompl =
    {SubjectVerbBe}({FuncQualifier}{WS}+)?{Complement}
```

   {FuncSubject} matches a subject function (*{*SUBJ[<>]? ... *SUBJ[<>]?}*), {VPBe} matches a verb which demands a nominative predicate and {NPAcc} and {NPDat} match an accusative and a dative noun phrase, respectively. {Complement} matches a sequence of nominative adjective/noun phrases or a sequence of past participle phrases.

   The above patterns match substrings in which the verb phrase appears to the left of the complement. Therefore, the markers *{*COMP< ... *COMP<}* are used. As before, string searches are used to find where the complement starts in the matched substring.

To illustrate, given the input substrings below:

- {*SUBJ> [NPn ég NP] *SUBJ>} [VPb var VPb] [APn lítill AP] (I was small)
- {*SUBJ> [NP saga NP] {*QUAL [NP [AP vísindalegrar AP] sálfræði NP] *QUAL} *SUBJ>} [VPb er VPb] [AdvP ekki AdvP] [AP [AdvP ýkja AdvP] löng AP]
- {*SUBJ> [NP [AP sálfræðileg AP] viðfangsefni NP] *SUBJ>} [VPb hafa verið VPb] [NP manninum NP] [AP hugleikin AP]

...the corresponding output is:

- {*SUBJ> [NPn ég NP] *SUBJ>} [VPb var VPb] {*COMP< [APn lítill AP] *COMP<}
- {*SUBJ> [NP saga NP] {*QUAL [NP [AP vísindalegrar AP] sálfræði NP] *QUAL} *SUBJ>} [VPb er VPb] [AdvP ekki AdvP] {*COMP< [AP [AdvP ýkja AdvP] löng AP] *COMP<}
- {*SUBJ> [NP [AP sálfræðileg AP] viðfangsefni NP] *SUBJ>} [VPb hafa verið VPb] [NP manninum NP] {*COMP< [AP hugleikin AP] *COMP<}

5. **Func_OBJ**: This transducer puts the markers *{*OBJ> ... *OBJ>}* or *{*OBJ< ... *OBJ<}* around direct objects of verbs. Refer to section 6.2.2.3 for a description of how objects should be annotated according to the annotation scheme.

   This transducer uses various patterns to recognise objects, depending on the exact sequence of subject-verb-object (SVO). The main order is SVO, but other sequences like VSO, OVS, and VO (no subject) are also recognised.

   Here, we only discuss the main case (SVO), i.e. when a subject appears to the left of the finite verb phrase and an object appears to the right of the verb phrase (other cases are implemented similarly). For this case, an adverb phrase or a preposition phrase can appear before the object.

```
RelCP = {OpenCP}~sem{WS}+{ConjTag}{CloseCP}
SubjectRelCP = {FuncSubject}|{RelCP}
```

```
SubjVerb = {SubjectRelCP}{WS}+{VP}{WS}+
Object = ({FuncQualifier}{WS}+)?
        ({NP}|{NPs}|{AP}|{APs})({WS}+{FuncQualifier})?

SubjVerbObj = {SubjVerb}{Object}
SubjVerbAdvPObj = {SubjVerb}{AdvPs}{Object}
SubjVerbPPObj = {SubjVerb}{AdvPs}?{PP}{WS}+{Object}
```

{SubjectRelCP} matches a subject function or a relativizer. {SubjVerb} consists of a subject and a finite verb phrase. An {Object} is a sequence of adjective/noun phrases, optionally prefixed or suffixed with a qualifier.

The above patterns match substrings in which the object appears to the right of the verb phrase. Therefore, the markers *{*OBJ< ... *OBJ<}* are used. As before, string searches are used to find where the object starts in the matched substring.

To exemplify, given the input substrings below:

- {*SUBJ> [NP faðmur NP] {*QUAL [NP hans NP] *QUAL} *SUBJ>} [VP umlykur VP] [NP [APs [AP lágreist AP] [AP svört AP] APs] húsin NP]
- [CP sem CP][VP upplýsti VP] {*QUAL [NP hennar NP] *QUAL} [NP líf NP] (which enlightened her life)
- {*SUBJ> [NP þau NP] *SUBJ>} [VP fundu VP] [MWE_AdvP hér og hvar MWE_AdvP] [NP [AP gamla AP] skápa NP]

...the corresponding output is:

- {*SUBJ> [NP faðmur NP] {*QUAL [NP hans NP] *QUAL} *SUBJ>} [VP umlykur VP] {*OBJ< [NP [APs [AP lágreist AP] [AP svört AP] APs] húsin NP] *OBJ<}
- [CP sem CP][VP upplýsti VP] {*OBJ< {*QUAL [NP hennar NP] *QUAL} [NP líf NP] *OBJ<}
- {*SUBJ> [NP þau NP] *SUBJ>} [VP fundu VP] [MWE_AdvP hér og hvar MWE_AdvP] {*OBJ< [NP [AP gamla lkfosf AP] skápa nkfo NP] *OBJ<}

6. **Func_OBJ2**: The purpose of this transducer is fourfold. First, it marks indirect objects of di-transitive verbs. Second, nominative adjective phrases and verb past participle phrases, which have not been assigned a function by the previous transducers, are marked as complements. Third, the transducer marks objects and complements of complements (refer to section 6.2.2.3) Lastly, the transducer marks nominative objects of verbs that demand oblique case subjects (refer to section 6.2.2.3).

   Here, we only describe patterns used for marking indirect objects. Recall, that the previous transducer in the sequence (Func_OBJ) has marked direct objects. In the case where this direct object is in the dative case and a accusative noun phrase follows, it is very likely that the assumed direct object is, in fact, an indirect object and the following noun phrase is the direct object. Thus, the sequence <verb, dative noun phrase, accusative noun phrase> is a strong indication of the sequence <verb, indirect object, direct object>.

```
NPObj = "{*OBJ<"{WS}+{DatObj}{WS}+"*OBJ<}"
DatObj = {NPDat} | {NPsDat}

VerbObjObj = ({VP}|{VPInf}){WS}+{NPObj}{WS}+
                ({PP}{WS}+)?({NPAcc}|{NPsAcc})
```

   The patterns {VP} and {VPInf} match a finite verb phrase or an infinitive verb phrase, respectively. The {NPObj} pattern matches a dative noun phrase(s) which has been marked with an object function. The {NPAcc} and {NPsAcc} match a accusative noun phrase or a sequence of accusative noun phrases, respectively.

   When finding a substring matching the {VerbObjObj} pattern, the transducer must therefore change the direct object marking to indirect and mark the noun phrase that follows as a direct object. This is accomplished with string searches/replacement.

   To exemplify, given the input substring: [VPi að cn segja sng VPi] {*OBJ< [NPd þér fp2eþ NP] *OBJ<} [NPa það fpheo NP], the resulting string is [VPi að cn segja sng VPi] {*IOBJ< [NPd þér fp2eþ NP] *IOBJ<} {*OBJ< [NPa það fpheo NP] *OBJ<}.

7. **Func_OBJ3**: This transducer marks dative objects of complement adjective phrases.

```
DatObj = {NPDat} | {NPsDat}
AdjCompl = {OpenComp}{WS}+{OpenAP}~{CloseAP}
           {WS}+{CloseComp}
ObjDat = {DatObj}({WS}+{FuncQualifier})?
ObjDatCompl = {ObjDat}{WS}+{AdjCompl}
ComplObjDat = {AdjCompl}{WS}+{ObjDat}
```

   The {NPDat} and {NPsDat} patterns match dative single noun phrases or a sequence of dative noun phrases, respectively. The {OpenComp} pattern matches an opening complement function marker and {OpenAP} the opening adjective phrase marker. Hence, the {ObjDatCompl} and {ComplObjDat} patterns match dative objects that either precede or follow an complement adjective phrase.

   Additionally, this transducer marks accusative noun phrases which have not yet received a function label and have a specific format, as temporal expressions. An example of such phrase is the accusative noun phrase *[NP eitt vor NP]* (one spring).

8. **Func_SUBJ2**: The purpose of this transducer is simply to annotate "stand-alone" nominative noun phrases that have not been assigned a function by the previous transducers. Each such noun phrase receives the markers *{*SUBJ ... *SUBJ}*.

C.3 Other transducers

1. **Clean2**: Recall that the **Case_AP** and **Case_NP** transducers add case information to the AP and NP labels (to use for other transducers that run later in the sequence). This information is removed by this transducer. Additionally, this transducer suppresses two or more whites paces into one.

```
PhraseCase = [nadg]     /* nom, acc, dat, gen */
StartNP = {OpenNP}{PhraseCase}
StartAP = {OpenAP}{PhraseCase}
```

```
%%

^" "          {;}
{WS}+    { System.out.print(" ");}
{StartNP}    { System.out.print("[NP");}
{StartAP}    { System.out.print("[AP");}
```

C.4 Multiword expressions

This section shows a list of MWEs recognised by our shallow parser, *IceParser*. The list is in the form of regular expression and is obtained from the three finite-state transducers responsible for annotating the MWEs.

Transducer Phrase_MWEP1

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

MWE_PP = [fF]yrir{PrepPart}(aftan|austan|framan|handan|neðan|
         norðan|ofan|sunnan|utan|vestan){AdverbPart}
```

Transducer Phrase_MWEP2

```
MainPrepPart = (að|af|á|eftir|frá|fyrir|hjá|í|með|til|um|úr|
                við|yfir|undir){PrepPart}

MWE_PP = [aA]ft(an|ur){AdverbPart}{MainPrepPart}        |
    [aA]ust(an|ur){AdverbPart}{MainPrepPart}            |
    [áÁ]{AdverbPart}(eftir|meðal|milli|móti|undan){PrepPart}|
    [bB]ak{NounPart}við{PrepPart}                       |
    [fF]ram(an|mi)?{AdverbPart}{MainPrepPart}           |
    [gG]egn{AdverbPart}um{PrepPart}                     |
    [hH]ér{AdverbPart}(á|fyrir|hjá|í|við|undir){PrepPart} |
    [hH]andan{AdverbPart}(að|af|frá|fyrir|í|um|við|yfir)
             {PrepPart}  |
    [nN]eðan{AdverbPart}{MainPrepPart}                  |
    [nN]ið(ur|ri){AdverbPart}{MainPrepPart}             |
```

```
    [nN]orð(an|ur){AdverbPart}{MainPrepPart}          |
    [iI]nn(i|an)?{AdverbPart}{MainPrepPart}           |
    [íÍ]{AdverbPart}(gegnum|kringum){PrepPart}        |
    [oO]fan{AdverbPart}{MainPrepPart}                 |
    [rR]étt{AdverbPart}(í|á|hjá|við){PrepPart}        |
    [sS](unnan|uður){AdverbPart}{MainPrepPart}        |
    [uU]pp{AdverbPart}{MainPrepPart}                  |
    [uU]ppi{AdverbPart}(á|í){PrepPart}                |
    [uU]tan{AdverbPart}{MainPrepPart}                 |
    [úŰ]t{AdverbPart}{MainPrepPart}                   |
    [úŰ]ti{AdverbPart}(á|í|við){PrepPart}             |
    [vV]esta(an|ur){AdverbPart}{MainPrepPart}         |
    [yY]fir{AdverbPart}{MainPrepPart}                 |
    [þÞ]rátt{AdverbPart}fyrir{PrepPart}
```

Transducer Phrase_MWE

```
Ada = {WS}*að{WS}+{AdverbTag}
Adc = {WS}*að{WS}+{ConjTag}
Adi = {WS}*að{WS}+{InfinitiveTag}
Adp = {WS}*að{WS}+{PrepTag}
Af = {WS}*[aA]f{WS}+{PrepTag}
Afa = {WS}*[aA]f{WS}+{AdverbTag}
Aa = {WS}*á{WS}+{AdverbTag}
Ap = {WS}*á{WS}+{PrepTag}
Adeins = {WS}*aðeins{WS}+{AdverbTag}
Adur = {WS}*áður{WS}+{AdverbTag}
Aftur = {WS}*[aA]ftur{WS}+{AdverbTag}
Afram = {WS}*áfram{WS}+{AdverbTag}
Alls = {WS}*[aA]lls{WS}+{PronounTag}
Allt = {WS}*[aA]llt{WS}+{PronounTag}
Alveg = {WS}*[aA]lveg{WS}+{AdverbTag}
Annad = {WS}*annað{WS}+{PronounTag}
Annars = {WS}*[aA]nnars{WS}+{PronounTag}
Aukp = {WS}*auk{WS}+{PrepTag}
Auki = {WS}*auki{WS}+{AdverbTag}
An = {WS}*[áÁ]n{WS}+{PrepTag}
Bak = {WS}*bak{WS}+{NounTag}
Badum = {WS}*[bB]áðum?{WS}+{PronounTag}
```

```
Beggja = {WS}*[bB]eggja{WS}+{PronounTag}
Betur = {WS}*betur{WS}+{AdverbTag}
Bil = {WS}*bil{WS}+{NounTag}
Blatt = {WS}*[bB]látt{WS}+{AdverbTag}
Boginn = {WS}*bóginn{WS}+{NounTag}
Daemis = {WS}*dæmis{WS}+{NounTag}
Ed = {WS}*eð{WS}+{ConjTag}
Eda = {WS}*[eE]ða{WS}+{ConjTag}
Ef = {WS}*[eE]f{WS}+{ConjTag}
Eftir = {WS}*eftir{WS}+{PrepTag}
Einhvern = {WS}*[eE]inhvern{WS}+{PronounTag}
Einhvers = {WS}*[eE]inhvers{WS}+{PronounTag}
Eins = {WS}*[eE]ins{WS}+{AdverbTag}
Einsp = {WS}*[eE]ins{WS}+{PronounTag}
En = {WS}*en{WS}+{ConjTag}
Enn = {WS}*enn{WS}+{AdverbTag}
Enda = {WS}*[eE]nda{WS}+{AdverbTag}
Engan = {WS}*[eE]ngan{WS}+{PronounTag}
Engu = {WS}*[eE]ngu{WS}+{PronounTag}
Einu = {WS}*einu{WS}+{AdjectiveTag}
Einun = {WS}*einu{WS}+{NumeralTag}
Er = {WS}*er{WS}+{VerbFiniteTag}
Fer = {WS}*fer{WS}+{VerbFiniteTag}
Ferns = {WS}*[fF]erns{WS}+{AdjectiveTag}
Fraa = {WS}*frá{WS}+{AdverbTag}
Fram = {WS}*[fF]ram{WS}+{AdverbTag}
Framvegis = {WS}*framvegis{WS}+{AdverbTag}
Fremst = {WS}*fremst{WS}+{AdverbTag}
Fyrr = {WS}*[fF]yrr{WS}+{AdverbTag}
Fyrst = {WS}*[fF]yrst{WS}+{AdverbTag}
Hattar = {WS}*háttar{WS}+{NounTag}
Heldur = {WS}*[hH]eldur{WS}+{AdverbTag}
Her = {WS}*[hH]ér{WS}+{AdverbTag}
Herna = {WS}*[hH]érna{WS}+{AdverbTag}
Hinn = {WS}*[hH]inn{WS}+{PronounTag}
Hins = {WS}*[hH]ins{WS}+{PronounTag}
Hinum = {WS}*[hH]inum?{WS}+{PronounTag}
Hvad = {WS}*[hH]það{WS}+{PronounTag}
```

```
Hvar = {WS}*[hH]var{WS}+{AdverbTag}
Hvort = {WS}*hvort{WS}+({ConjTag}|{PronounTag})
Hverju = {WS}*hverju{WS}+{PronounTag}
Hvers = {WS}*[hH]vers{WS}+{PronounTag}
Haegra = {WS}*[hH]ægra{WS}+{AdjectiveTag}
I = {WS}*í{WS}+{PrepTag}
Jafnt = {WS}*[jJ]afnt{WS}+{AdverbTag}
Jafnvel = {WS}*[jJ]afnvel{WS}+{AdverbTag}
Konar = {WS}*konar{WS}+{NounTag}
Kosti = {WS}*kosti{WS}+{NounTag}
Kyns = {WS}*kyns{WS}+{NounTag}
Lagi = {WS}*lagi{WS}+{NounTag}
Leid = {WS}*leið{WS}+{NounTag}
Leyti = {WS}*leyti{WS}+{NounTag}
Likt = {WS}*[lL]íkt{WS}+{AdverbTag}
Margs = {WS}*[mM]args{WS}+{AdjectiveTag}
Medal = {WS}*meðan{WS}+{AdverbTag}
Medan = {WS}*meðan{WS}+{ConjTag}
Megin = {WS}*megin{WS}+{AdverbTag}
Meira = {WS}*[mM]eira{WS}+{AdjectiveTag}
Mikid = {WS}*mikið{WS}+{AdverbTag}
Minnsta = {WS}*minnsta{WS}+{AdjectiveTag}
Min = {WS}*[mM]ín{WS}+{PronounTag}
Moti = {WS}*móti{WS}+{NounTag}
Mynda = {WS}*mynda{WS}+{VerbInfinitiveTag}
Neins = {WS}*[nN]eins{WS}+{PronounTag}
Nokkru = {WS}*[nN]okkru{WS}+{PronounTag}
Nokkurn = {WS}*[nN]okkurn{WS}+{PronounTag}
Nokkurs = {WS}*[nN]okkurs{WS}+{PronounTag}
Ny = {WS}*ný{WS}+{AdverbTag}
Nyju = {WS}*nýju{WS}+{AdjectiveTag}
Rettu = {WS}*[rR]éttu{WS}+{AdjectiveTag}
Sagt = {WS}*sagt{WS}+{VerbTag}
Sama = {WS}*[sS]ama{WS}+{PronounTag}
Saman = {WS}*saman{WS}+{AdverbTag}
Sams = {WS}*[sS]ams{WS}+{PronounTag}
Samt = {WS}*[sS]amt{WS}+{AdverbTag}
Segja = {WS}*segja{WS}+{VerbInfinitiveTag}
```

```
Sem = {WS}*[sS]em{WS}+{ConjTag}
Sema = {WS}*sem{WS}+{AdverbTag}
Sidur = {WS}*síður{WS}+{AdverbTag}
Sin = {WS}*[sS]ín{WS}+{PronounTag}
Sinni = {WS}*sinni{WS}+{NounTag}
Sjalfsogdu = {WS}*sjálfsögðu{WS}+{AdjectiveTag}
Smam = {WS}*[sS]mám{WS}+{AdverbTag}
Stad = {WS}*stað{WS}+{NounTag}
Stadar = {WS}*staðar{WS}+{NounTag}
Stundum = {WS}*stundum{WS}+{AdverbTag}
Svo = {WS}*svo{WS}+{AdverbTag}
Og = {WS}*[oO]g{WS}+{ConjTag}
Tila = {WS}*[tT]il{WS}+{AdverbTag}
Tilp = {WS}*[tT]il{WS}+{PrepTag}
Tvenns = {WS}*[tT]venns{WS}+{AdjectiveTag}
Um = {WS}*[uU]m{WS}+{PrepTag}
Ur = {WS}*[úŰ]r{WS}+{PrepTag}
Vegar = {WS}*vegar{WS}+{NounTag}
Veginn = {WS}*veginn{WS}+{NounTag}
Vegna = {WS}*vegna{WS}+{PrepTag}
Vid = {WS}*[vV]ið{WS}+{AdverbTag}
Vill = {WS}*vill{WS}+{VerbFiniteTag}
Vinstra = {WS}*[vV]instra{WS}+{AdjectiveTag}
Visu = {WS}*vísu{WS}+{AdjectiveTag}
Ymiss = {WS}*[ýÝ]miss{WS}+{PronounTag}
Thad = {WS}*það{WS}+{PronounTag}
Thar = {WS}*[þÞ]ar{WS}+{AdverbTag}
Theim = {WS}*[þÞ]eim{WS}+{PronounTag}
Thess = {WS}*þess{WS}+{PronounTag}
Thett = {WS}*þétt{WS}+{AdverbTag}
Thin = {WS}*[þÞ]ín{WS}+{PronounTag}
Tho = {WS}*[þÞ]ó{WS}+{AdverbTag}
Thott = {WS}*[þÞ]ótt{WS}+{ConjTag}
Threnns = {WS}*[þÞ]renns{WS}+{AdjectiveTag}
Thvi = {WS}*því{WS}+{PronounTag}
Thvia = {WS}*því{WS}+{AdverbTag}
Odru = {WS}*[öÖ]ðru{WS}+{PronounTag}
Ofugu = {WS}*[öÖ]fugu{WS}+{AdjectiveTag}
```

```
Ollu = {WS}*[öÖ]llu{WS}+{PronounTag}

MWE_AdvP = {Adp}({Nyju}|{Sjalfsogdu}|{Visu}|
                {Minnsta}{Kosti}|({Nokkru}|{Odru}){Leyti}) |
        ({Alls}|{Annars}|{Einhvers}){Stadar}|
        {Afa}{Og}{Tila}                  |
        {Aftur}{Ap}({Bak}|{Moti})        |
        {Aftur}{Og}{Aftur}               |
        {Allt}{I}({Einu}|{Lagi})         |
        {Allt}{Adp}{Thvi}                |
        ({Annars}|{Hins}){Vegar}         |
        {Aukp}{Thess}                    |
        {Aa}({Ny}|{Stundum})             |
        {Ap}{Hinn}{Boginn}               |
        {Adur}{Fyrr}                     |
        {An}{Thess}{Adc}                 |
        {Blatt}{Afram}                   |
        ({Badum}|{Beggja}|{Herna}|{Hinum}|{Haegra}|{Min} |
        {Rettu}|{Sin}|{Thin}|{Vinstra}|{Theim}|{Odru}|{Ofugu}){Megin}|
        ({Einhvern}|{Engan}|{Nokkurn}){Veginn} |
        {Eda}{Ollu}{Heldur}              |
        {Ef}{Tila}{Vill}                 |
        {Ekki}{Sist}                     |
        {Engu}{Ada}{Sidur}               |
        {Enn}?{Einun}{Sinni}             |
        {Fram}{Og}{Aftur}                |
        {Fyrst}{Og}{Fremst}              |
        {Her}{Og}({Thar}|{Hvar}|{Nu})    |
        {Hvar}{Sem}{Er}                  |
        {Hvort}({Ed}|{Sem}){Er}?         |
        {Hvers}{Vegna}                   |
        {Haerra}{Og}{Haerra}             |
        {Jafnt}{Og}{Thett}               |
        {Meira}{Adi}?{Segja}             |
        {Nokkru}{Sinni}                  |
        {Og}{Svo}{Framvegis}             |
        {Sama}{Hvort}                    |
        {Samt}{Sem}{Adur}                |
```

```
        {Sem}({Sagt}|{Betur}{Fer})      |
        {Sidur}{En}{Svo}                |
        {Smam}{Saman}                   |
        {Svo}{Og}                       |
        {Tilp}({Daemis}|{Adi}{Mynda})   |
        {Tila}{Og}{Fraa}                |
        {Um}({Leid}|{Thad}{Bil})        |
        {Vegna}{Thess}                  |
        {Vid}{Og}{Vid}                  |
        {Thess}({Vegna}|{I}{Stad})      |
        {Thar}?{Ada}{Auki}              |
        {Thar}{Aa}{Medal}               |
        {Odru}{Hverju}

MWE_AP = ({Alls}|{Annars}|{Einhvers}|{Einsp}|{Ferns}|{Hvers}|
        {Margs}|{Neins}|{Nokkurs}|{Sams}|{Tvenns}|{Threnns}|
        {Ymiss}|{Thess}){Konar} |
        {Hvers}{Kyns}   |
        {Thess}{Hattar}

MWE_CP = ({Af}|{Ur})?{Thvi}{Adc}    |
        {Aa}{Medan}                 |
        {Eftir}{Adc}                |
        {Adur}{En}                  |
        {Alveg}?({Eins}|{Likt}){Og} |
        ({Enda}|{Jafnvel}){Thott}   |
        {Hvorki}{Meiraa}{Ne}{Minna}{En} |
        {Svo}{Adc}                  |
        {Svo}{Mikid}?{Sem}          |
        ({Tilp}|{An}){Thess}?({Adi}|{Adc}) |
        {Um}{Leid}{Og}              |
        {Vegna}{Thess}{Adc}         |
        {Thar}({Tila}|{Sem})        |
        {Tho}{Adc}                  |
        {Thvia}{Adeins}?{Adc}
```