Summary

Finite-state automata are models that consist of a finite number of states and transitions between states. Although it is known that these models, featuring a restricted amount of memory, are in principle incapable of deep analysis of natural language (as carried out by wide coverage parsers), certain features still render them very much of use in syntactic analysis. Finite-state automata are efficient, good at representing language locally, and there exist effective methods for deriving them from a collection of data. This dissertation describes how finite-state techniques can be used effectively in syntactic analysis.

In particular, this dissertation constitutes empirical research into the construction and use of a finite-state approximation of a wide-coverage parser to increase parsing performance. The finite-state approximation is in the form of a hidden Markov model, inferred from parser-annotated data. This model is used in a part-of-speech tagger, which is applied in various ways and using several different models to reduce ambiguity in parsing, by setting it up as a filter that removes unlikely options in the first stage of parsing.

In chapter 1 the problems of ambiguity and inefficiency in wide-coverage computational language processing are introduced. The approach taken in this work to reduce ambiguity and thus increase efficiency, as summarized above, is described, and an overview of the rest of the thesis is presented.

Chapter 2 defines the finite-state automaton, both in its deterministic and non-deterministic form, and the weighted and non-weighted variants. Reasons for using finite-state techniques in language processing are presented. The first reason is the linear complexity associated with finite-state techniques. Second, a finite-state automaton defining a language can be minimized and determinized resulting in a compact automaton that represents the same language. Third, the class of regular languages, which is the class of languages defined by finite-state automata, is closed under the operations of union, intersection, Kleene star, concatenation and complementation.

Motivation for using finite-state models of syntax is given. The syntactic phenomena of center-embedding and cross-serial dependencies are both non-
regular, yet cases of these constructions are only successfully processed by humans as long as a certain small level of recursion is not exceeded. This suggests that human linguistic competence is more powerful than finite-state, yet linguistic performance is not. In this context, three aspects of human language processing are presented suggesting that human language performance is finite-state. First, humans have a finite amount of working memory available. Second, humans have problems processing cases of center-embedding and cross-serial dependencies, which are non-regular constructions. Third, human language processing efficiency is reminiscent of finite-state efficiency.

The chapter then turns to a description of different approaches to finite-state approximation of more powerful descriptions of language syntax. The methods are approximation through RTNs, approximation through grammar transformation, approximation through the use of a restricted stack, and approximation using n-gram models. The methods all expect the target of approximation to be a context-free grammar. However, in systems using stochastic attribute-value grammars, such as the Alpino system which is the target of approximation in this research, features that cannot be described by a context-free grammar are used. In addition, components to deal with real-life parsing problems such as unknown words should preferably be part of the approximation. Taking these points into account, the method of inference, to be described below, is preferred.

In chapter 3, it is shown how a stochastic finite-state model can be derived from a wide-coverage parser through inference. In later chapters, a number of different models constructed using this technique will be used in the task of reducing lexical and structural ambiguity in parsing.

Grammatical inference consists of deducing a grammar from a sample of language. Its goal is to find a set of grammatical rules, or equivalently an automaton, that models the patterns present in the example sentences.

Based on work by Gold [42] it is concluded that a language can be learned from a stochastic sample of that language. A type of finite-state model derived from a stochastic sample through inference is the n-gram model, in which the probability of a certain observation (such as a particular word occurring) is based on the previous $n - 1$ observations. This general idea can be applied in various ways leading to models that differ in complexity.

We first introduce the Markov model. The Markov model is a stochastic finite-state automaton in which the states directly represent the observations. In a typical Markov model of natural language, the states directly represent the words.

Then the more complex hidden Markov model (HMM) is presented. The HMM is a stochastic finite-state automaton in which both state transitions as well as the production of output symbols are governed by probability
distributions. In a typical HMM of natural language, the states represent part-of-speech tags while the output symbols are the words. This particular kind of HMM is known as the POS tagging model. The POS tagging model can be used to define the best POS tag sequence for a given sequence of words based on the probabilities of word-tag pairs as well as probabilities of tag n-grams.

In constructing a HMM through inference, the probabilities of word-tag combinations and tag n-grams are estimated from the frequencies of these observations in a training corpus.

In chapter 4 it is shown how a POS tagging HMM, inferred from data annotated with lexical categories by the parser, is used to increase the efficiency and accuracy of the same parser. (Thus, the parser’s lexical categories are treated as POS tags; in doing so, subcategorization information in the lexical categories is ignored in order to reduce the size of the resulting POS tag set.) The inferred HMM combines unigram, bigram, and trigram data through linear interpolation. The model is used in a POS tagger which operates as a POS tag filter during the parser’s lexical analysis phase to reduce the number of lexical categories assigned to each word in a sentence.

To every possible tag for a given word in the sentence, the POS tag filter assigns a value representing the probability of that tag. This value is composed of the tag’s forward and backward probabilities as defined the forward-backward algorithm for HMM training. Once every tag at a given position in the sentence has been assigned a score, tags that score considerably worse than the best tag at that position can be removed. A constant value is used to decide how much worse a tag’s score should be compared to the best score at that position for the tag to be removed. Increasing and decreasing this threshold value will lead to fewer and more tags being removed respectively.

The POS tag filter is trained on a newspaper corpus that has been annotated by the parser with the lexical categories as used in what the parser considered to be the best analyses for those sentences. The amount of training data is about 24 million words. The parser is used both with and without the filter to parse 220 sentences. Without the use of the filter, this takes an average time of 53 seconds per sentence, and results in an accuracy of 83%. With the filter, parsing time is reduced to an average of 14 seconds, and accuracy is increased to 85%.

In chapter 5 the POS tagging model of chapter 4 is extended with specific syntactic information in order to reduce the number of errors made by the tagger. Stand-alone tagging experiments show that the most frequent error is of a type that could often be avoided if the tagger was not restricted to a context of just two preceding tags. This particular error consists of mixing up the infinitive verb and the third person plural finite verb, for which Dutch
Summary

uses the same word form. This error could often be avoided if the tagger had access to the information whether or not a finite verb has already occurred in the current clause, since if it has, this would rule out a finite verb at the current location.

Increasing the model’s reach by using n-grams longer than three words is not a solution: as the n-grams get longer, they become increasingly rare in training data, making it harder to correctly estimate their probabilities from their frequencies. Instead, the model is extended with a single feature in every state of the model that says whether or not a finite verb occurred in the current clause. The model is trained on data that has been annotated by the parser not only with POS tags, but also with values for this new binary feature.

The extended model is tested in two POS tagging experiments. In each of the experiments a different data set is used: first, a large amount of newspaper text annotated by the Alpino parser; second, a much smaller set of data which is the written part of the Eindhoven corpus, annotated with the Wotan tagset. When using the extended model, the number of errors where the infinitive verb is mixed up with the finite verb decreases considerably, by 66% and 37% respectively. Considering all types of errors, the error rate drops with 4.2% and 2.6% respectively. The differences between the improvements in the two experiments are explained by differences between the sets of data used.

Finally, the extended model is used in the POS tag filter described in chapter 4 to increase parsing performance. The results in terms of parser efficiency and accuracy are compared to the results when using the standard model, showing only a small increase in efficiency. This is explained as follows. Using the standard model, the POS tag filter, even when it confuses the two verb forms, will often supply the parser with both options, after which the parser typically selects the correct tag. Using the extended model, the tagger will more often remove the wrong tag; this leads to a small increase in efficiency as the parser will not spend time in considering the incorrect alternative, but accuracy remains the same.

In chapter 6, the ideas from chapters 4 and 5 are combined in creating a model of syntactic structure on the level of chunks, or minimal phrases. As in chapter 4, a filter is based on the model that aims at reducing ambiguity in parsing, but here this concerns structural ambiguity. Using a POS tagging HMM in which information of the level of chunk structures has been incorporated, brackets can be placed in a sentence that reflect chunk structure. The bracketed sentence is used as input to the parser, which during parsing is forced to have phrases start and end at the positions of opening and closing brackets respectively, thus reducing the number of possible analyses.
The tagger is modified so that it not only assigns POS tags but chunk tags as well. Using a tagger to assign chunk tags is known as chunking as tagging. Typically a set of three labels is used that specify for every word in the sentence whether that word is at the start, at the end, inside, or outside of a chunk. After explaining this approach, different methods of assigning both chunk tags and POS tags are presented. A point on which these methods differ is whether chunk tags and POS tags are assigned simultaneously, as opposed to assigning the chunk tags after the POS tags. Another difference lies in the nature of the chunk tags, which can consist of just the chunk labels as described above, or of a concatenation of chunk labels and POS tags. These differences lead to four possible chunking as tagging methods. Once each word has received a chunk tag, these can be straightforwardly converted into chunk brackets that represent the start and end of a chunk.

The experiments carried out only involve noun phrase chunks or baseNPs. In order to evaluate the tagger, first a stand-alone chunking experiment is run on a standard dataset for chunking, consisting of a part of the Wall Street Journal corpus. The four different approaches to chunking as tagging are used, and the results in terms of precision and recall in recognizing baseNPs are compared. The best result is attained using the method that assigns chunk tags after POS tags have already been assigned, and which uses chunk tags that are concatenations of chunk labels and POS tags.

Then, the POS tagging model that is extended with chunk information is applied as a filter as described above, in order to reduce structural ambiguity for the parser. Since the parser does not work with baseNPs, a model is now created that concerns innermost NPs, or noun phrases that do not contain other noun phrases. Training data is approximately 9 million words of newspaper text, annotated with POS tags and innermost NP brackets by the parser. A set of 500 sentences is used as input to the parser, both with and without brackets as assigned to them using the chunk tagger. For comparison the same set of sentences is also parsed when all correct brackets are already present. Although the use of the correct brackets leads to an improvement in the performance of the parser, showing that this type of information can in principle be beneficial, the results of the other experiments show that the use of the brackets as assigned by the chunk tagger in its current implementation does not have a positive effect on parser efficiency and accuracy. At the end of the chapter, a number of suggestions for improving this method are given.

In chapter 7 the conclusions drawn from the previous chapters are presented. Summarizing, it can be stated that a finite-state approximation of a parser derived through inference from a corpus annotated by the parser can be used successfully to increase the efficiency and accuracy of the parser.

A large increase in parsing efficiency results from applying the approxima-
tion in a POS tag filter that is used to reduce the number of lexical categories assigned by the parser to each word in a sentence during lexical analysis. At the same time, parsing accuracy is increased.

Extending the approximation with a binary feature, which in stand-alone tagging experiments helped to considerably reduce the number of occurrences of the tagger's most frequent type of error, results in an additional but small increase in parsing efficiency when used in the POS tag filter.

Using inference to create an approximation of the parser that not only represents the parser in its preferences for assigning lexical categories but also with respect to sentence structure on the level of chunks, did not, in its current implementation, lead to an increase in parser performance, although experiments show that in principal this kind of information can be beneficial.

The use of the method of inference from parser-annotated data to arrive at a finite-state approximation of the parser means that the methods described above are not restricted to being used with a specific grammatical framework; the internal functioning of the parser is irrelevant as the approximation is derived indirectly through annotated data. A further advantage of this method is that it is automatic; the approximation itself does not need to be manually constructed, thus a new approximation can be straightforwardly created if the parser has been modified, and the annotation of training data is done by the parser and not by humans, so that it is possible to create large amounts of training data.